E.T.S. de Ingeniería Industrial, Informática y de Telecomunicación

Algoritmo Deep Q-Learning para el aprendizaje por refuerzo de una estrategia de conducción en 2D

Grado en Ingeniería Informática

Trabajo Fin de Grado

Alumno: Mikel Vallejo del Moral

Director: Miguel Pagola Barrio

Pamplona, 2 de septiembre del 2021

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Índice

Índice de ilustraciones

1. Introducción

a. Resumen del trabajo

El campo de la inteligencia artificial es relativamente joven y ha tenido un desarrollo irregular a lo largo de los años con periodos que recuerdan a la comparación de la montaña rusa. Sin embargo, en la actualidad se está viviendo uno de los momentos de mayor crecimiento. Esta elevación se produce gracias a diversos factores que se han juntado en el mismo intervalo de tiempo como puede ser; el manejo de grandes masas de datos, la procesamiento gráfico y tensorial, etc.

El progreso tecnológico está siendo en gran parte potenciado por la inteligencia artificial. Se puede ver a este campo del conocimiento como una rama del desarrollo tecnológico que se divide a su vez en más ramas que siguen este proceso de división. Dentro de esta, se encuentra el aprendizaje automático, que contiene al aprendizaje por refuerzo.

El aprendizaje por refuerzo es un área inspirada en la psicología conductista y es utilizado para determinar qué acciones debe escoger un agente en un entorno dado con el fin de maximizar una recompensa acumulada. A través del ensayo y error, este es capaz de aprender una política ante ese mismo entorno, donde el objetivo es conseguir que esta política sea óptima.

El análisis realizado se enfocado en la aplicación de este tipo de aprendizaje combinado con redes neuronales profundas a través del algoritmo "Deep Q Learning". Este algoritmo es es una mejora del original, el algoritmo "Q-learning". Este, a diferencia de su antecesor, es capaz de utilizar redes neuronales para calcular las acciones del agente. Un breve resumen del funcionamiento podría ser el siguiente: el agente va realizando acciones y mediante las recompensas y estados que se recogen se va alimentado a la red neuronal, que procesa esta información y devuelve el valor Q de todas las posibles acciones de salida.

En el aspecto relacionado con el desarrollo de las redes neuronales se ha empleado un paquete de Python diseñado para realizar cálculos numéricos haciendo uso de la programación de tensores mediante el paquete de PyTorch, el cual permite su ejecución en GPU, lo que acelera los cálculos.

Finalmente, los resultados de la experimentación realizada han sido evaluados según los criterios ofrecidos por el distribuidor de entornos utilizado (OpenAI Gym), sacando conclusiones de este tipo de algoritmo combinado con las herramientas utilizadas. Se han propuesto varias opciones de líneas futuras relacionadas con el trabajo realizado y se han comentado los diferentes problemas que han ido surgiendo a lo largo de todo proyecto.

b. Motivación

La naturaleza del aprendizaje es la interacción con el entorno que nos rodea. A medida que qué comprobamos los efectos que se producen en él a través de nuestras acciones, el conocimiento que teníamos se ve enriquecido. La causalidad generada es la guía hacia el objetivo que teníamos marcado desde el principio. Esta relación nos acompaña a lo largo de nuestras vidas, ya sea para aprender un idioma, para aprender a entablar una conversación formal, etc. Es por ello por lo que el aprendizaje tiene una conexión robusta con la interacción, siendo esta uno de los pilares de la mayor parte de las teorías de aprendizaje e inteligencia artificial.

En este trabajo de investigación se utilizará una combinación de algoritmos de aprendizaje computacional. Se tratará de combinar el aprendizaje por refuerzo, a través del algoritmo Q-Learning, que está enfocado en el aprendizaje dirigido por objetivos mediante la interacción con el entorno y, se aprovechará el funcionamiento de las redes neuronales profundas para así de esta manera poder combinarlos.

El objetivo planteado en este trabajo es utilizar el algoritmo Deep Q-Leaming para lograr el aprendizaje de una estrategia de conducción en 2D del entorno "CarRacing-v0" dentro del apartado Box2D de OpenAI Gym, un conjunto de herramientas para desarrollar y comparar algoritmos de aprendizaje por refuerzo. Como sugiere el aprendizaje por refuerzo, este camino de aprendizaje se realizará sin que al agente se le indiquen parámetros fijos en el aprendizaje en relación con los datos del entorno, sino que el agente deberá aprender por su propia cuenta formalizado en la base de ensayo y error.

c. Palabras clave

- Aprendizaje por refuerzo
- Deep Q Learning
- Redes Neuronales
- PyTorch
- CarRacing

d. Objetivos

El objetivo final de este trabajo de investigación consiste en generar una simulación de una conducción automatizada de un coche en 2D a través del estudio de la rama del aprendizaje por refuerzo combinándola con el algoritmo Deep Q-Network. Para ello, una vez planteado el objetivo a conseguir, se descompone en el trabajo en diferentes objetivos más simples que den forma al final y de esta forma poder dividir la complejidad general.

El primer paso será aprender a desenvolverse con un nivel adecuado con las herramientas que se van a usar. Es decir, conseguir un cierto grado de conocimiento en; el lenguaje de programación Python junto con todas las librerías utilizadas (Pytorch, matplotlib, etc.), entornos seleccionados de OpenAI Gym, Anaconda Distribution junto con los procesos de instalación de dependencias...

Para poder solucionar de una mejor forma la problemática, se ha repartido el trabajo en diferentes objetivos que consiguen progresivamente una complejidad mayor. En un principio se prevé implementar el algoritmo propuesto en el entorno relacionado con los sistemas de control sencillos. Estos son más sencillos de trabajar que el entorno fijado como objetivo final del proyecto (CarRacing-v0), pero sirven para poder introducir y acomodar el algoritmo en el

proceso de aprendizaje. El entorno seleccionado como base de la implementación (CartPole-v0) servirá para poder empezar con el desarrollo del algoritmo y ampliar el conocimiento sobre el mismo.

Posteriormente, a través del algoritmo desarrollado en el entorno de sistemas de control sencillos, se entrenará el agente en el entorno objetivo con los respectivos cambios necesarios más las implementaciones añadidas que se requieran.

El objetivo principal del proyecto, por lo tanto, se enfocará en el desarrollo de un algoritmo (Deep Q-Learning) para alcanzar una estrategia de conducción en 2D en el entorno de CarRacing-v0.

2. Algoritmo Deep Q-Learning

En el presente capítulo se presentan diferentes conceptos teóricos utilizados en los algoritmos aplicados en el trabajo de investigación.

a. Aprendizaje por refuerzo

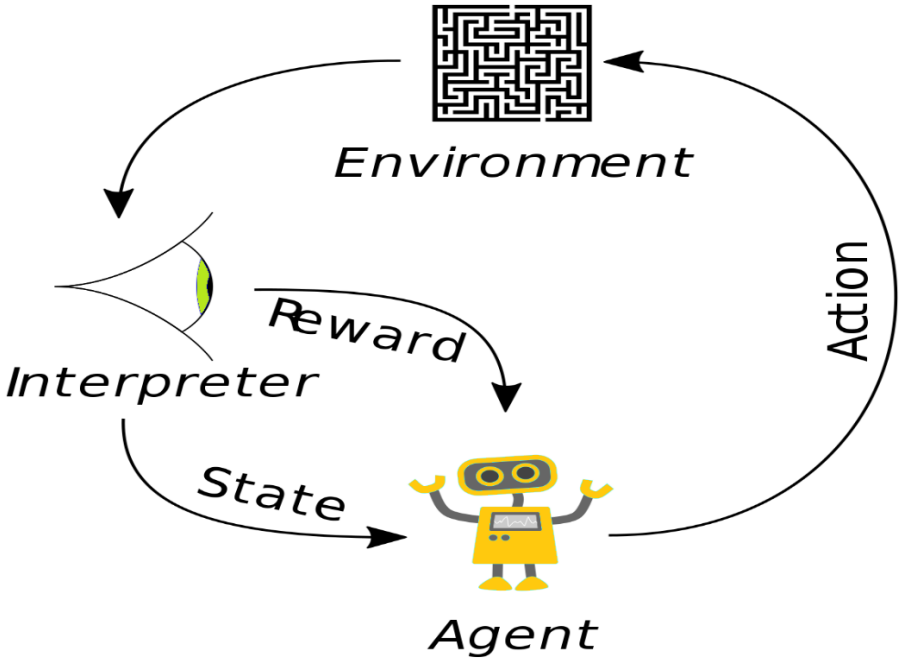
En este primer apartado se muestra de una forma breve y concisa los conceptos teóricos de aprendizaje por refuerzo que se han utilizado a lo largo del proyecto para la generación de los algoritmos.

Como ya se comentó en el primer capítulo, se conoce como aprendizaje por refuerzo [1] a una de las áreas del aprendizaje automático que utiliza como base la psicología conductista. En el proceso de aprendizaje una acción específica de un agente es seguida por una recompensa que puede ser positiva o negativa con el fin de enseñarle a este cómo elegir una acción que maximice la recompensa global del espacio de acción que tiene en disposición.

i. Introducción al AR

Para poder generar una visión más sencilla de entender, se puede apreciar en la *ilustración 1* los diferentes componentes que participan en un sistema de aprendizaje por refuerzo:

- **Agente:** es el modelo que se quiere entrenar. Aprende a tomar decisiones.
- **Ambiente/Entorno:** entorno dónde interactúa el agente.
- **Estado:** indicador o indicadores del estado del entorno.
- **Recompensa:**
    - Positiva: Si la acción realizada obtiene una bonificación.
    - Negativa: Si la acción realizada obtiene una penalización.
- **Acción:** posibles acciones que puede realizar el agente.



l

Ilustración 1. Diagrama de ejemplo de componentes del aprendizaje por refuerzo.

El proceso de aprendizaje se resume en una serie de pasos:

- El agente recibe un estado *s*, y una recompensa *r*
- El agente escoge una acción *a* del conjunto de acciones disponibles y la envía al entorno
- El entorno ejecuta la acción *a* y su estado pasa, entonces a *s*+1
- El intérprete recibe, entonces una observación *o*+1, interpreta el nuevo estado *s*+1 y calcula la recompensa *rt*+1 correspondiente a la transición (*s; a; s*+1)

Otro término importante es la política, que es la que determina la acción a realizar, dependiendo del estado en el que se encuentra el entorno. Se representa como un mapeado de estados a acciones, y que da la probabilidad de que el agente tome una acción determinada cuando el entorno se encuentra en un estado determinado. Finalmente, el agente debe aprender cuál es la política para obtener la mayor recompensa posible.

Adicionalmente, todo el comportamiento del sistema puede ser estocástico, lo que aumenta la dificultad del problema. Por ejemplo, si el sistema está en un estado determinado y realiza una acción, el estado siguiente resultante no tiene que ser siempre el mismo, sino seguir algún tipo de distribución aleatoria desconocida. El agente tiene entonces que aprender cuáles son las acciones que le pueden maximizar la recompensa esperada.

Además, el comportamiento del sistema suele dividirse en episodios en los que el entorno comienza en un estado inicial y termina con un estado final determinado, donde el intérprete determina también si un episodio está en proceso o ha terminado.

ii. Exploración y explotación

El primer concepto, la exploración, trata sobre saber qué acciones son las que se tienen que realizar para obtener una mayor recompensa en el proceso de aprendizaje. Como se ha comentado previamente, el sistema puede ser estocástico, por lo que cada acción es experimentada un número de veces para estimar el rendimiento al realizarla. Sin embargo, la explotación tiene como objetivo seleccionar una acción con el valor que optimice la etapa para el estado del agente, por ejemplo, seleccionar el valor de Q más alto en el algoritmo de Q-Learning.

Los problemas de aprendizaje por refuerzo se forman a partir de dos etapas. La primera de ellas es la etapa de predicción, donde dada una política se analiza el futuro. Y la segunda, la etapa de control, donde se optimiza ese futuro a través de escoger la mejor política posible.

iii. Objetivo del aprendizaje por refuerzo

Para que un agente aprenda a desarrollar estrategias que maximicen los beneficios, este tiene que interactuar con un entorno. Es un proceso de aprendizaje por condicionamiento similar a la realidad de un ser vivo, donde a través de la observación y la toma de decisiones en un entorno

se puede comprobar los efectos que se producen. Es decir, el objetivo en el aprendizaje por refuerzo es que el agente consiga aprender una política óptima para interactuar con el entorno.

Las aplicaciones del aprendizaje por refuerzo son muy variadas, desde sistemas de navegación en coches autónomos hasta recomendación sobre medicamentos a administrar a un paciente según su informe.

iv. Q-Learning

El estudio de este tipo de aprendizaje se realiza basándose en el concepto de procesos de decisión de Markov, los cuales definen una serie de condiciones que se deben cumplir en todo el sistema. Hay una gran variedad de algoritmos en el campo del aprendizaje por refuerzo, pero se ha puesto el foco del estudio en el algoritmo Q-Learning. Es un algoritmo sencillo pero eficiente en problemas con cierta complejidad y que se puede combinar con otros algoritmos de aprendizaje automático como las redes neuronales

El algoritmo Q-Learning [2] intenta aprender cuanta recompensa obtendrá a largo plazo para cada pareja de estados y acciones (s, a). A la función mencionada se le denomina como la función de acción-valor, la cual dentro del algoritmo Q-Learning se ve representada como Q(s,a). Esta función devuelve la recompensa que el agente recibirá al ejecutar la acción a desde el estado s, y asume que la política dictada por la función Q será la guía hasta el final del episodio.

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a')$$

Ilustración 2. Ecuación de Bellman

Para resolver el problema en cuestión, el algoritmo utiliza la ecuación de Bellman, que se utiliza para aprender los Q valores. El valor Q del estado de s y la acción a(Q(s,a)) debe ser igual a la recompensa r obtenida al ejecutar esa acción, más el valor Q de ejecutar la mejor acción posible a' desde el próximo estado s'. multiplicado por un factor de descuento γ (discount factor), que es un valor con rango $\gamma \in (0, 1]$. Este valor γ se usa para decidir cuánto peso le queremos dar a las recompensas a corto y a largo plazo, y es un hiperparámetro que debemos decidir nosotros.

b. Redes Neuronales

En el campo de la IA, las redes neuronales son una buena herramienta para poder aproximar funciones no lineales. Por lo tanto, una red neuronal para aproximar la función Q es una buena opción, y así se evita utilizar una tabla para representar la misma.

v. Introducción biológica

El cerebro del ser humano tiene una gran capacidad de procesamiento de datos gracias a las neuronas. Estas células se conectan entre ellas mediante impulsos eléctricos. Para conseguir esta comunicación una de ellas emite o no un impulso, todo en función de los estímulos de entrada que recibe.

El funcionamiento de las neuronas [3] está basado en recibir mediante sus dendritas señales de diversas neuronas a las que estás conectadas, y según las señales recibidas, emiten un impulso que se genera en el cuerpo celular y se transmite desde el axón de la propia neurona a otra que sea su vecina.

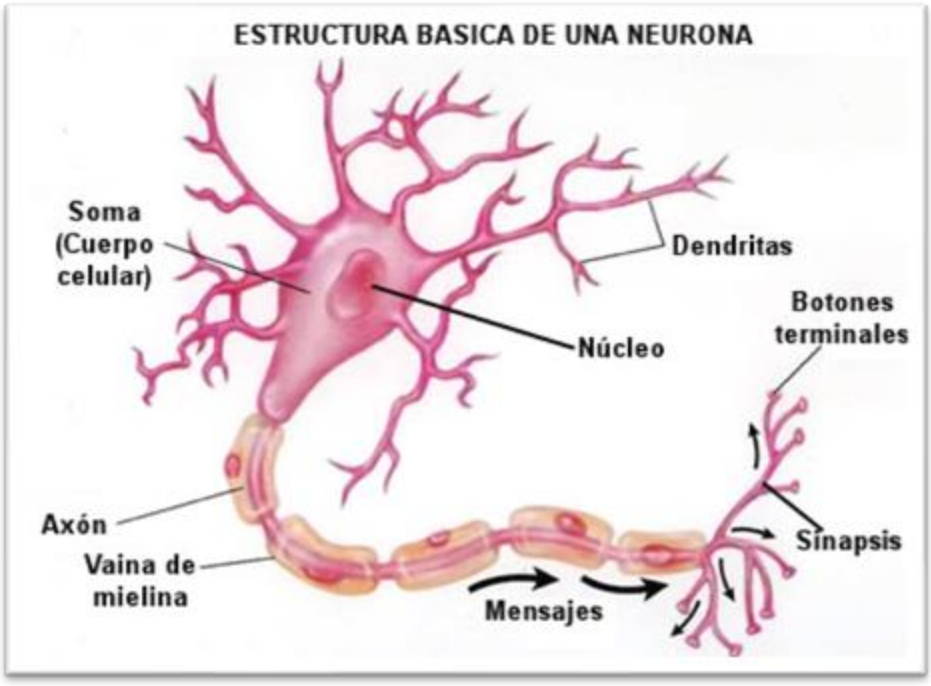


Ilustración 3. Estructura básica de una neurona - Esquema biológico

En este proceso de transmisión se genera un pulso que viaja de una neurona a otra que genera un salto sináptico, en el cual al llegar el impulso eléctrico al botón sináptico de la célula emisora, se separan las vesículas sinápticas, liberando así, neurotransmisores, que son recogidos por la neurona receptora a través de su dendrita dónde se genera en un pulso eléctrico que se propaga por el cuerpo celular de esta.

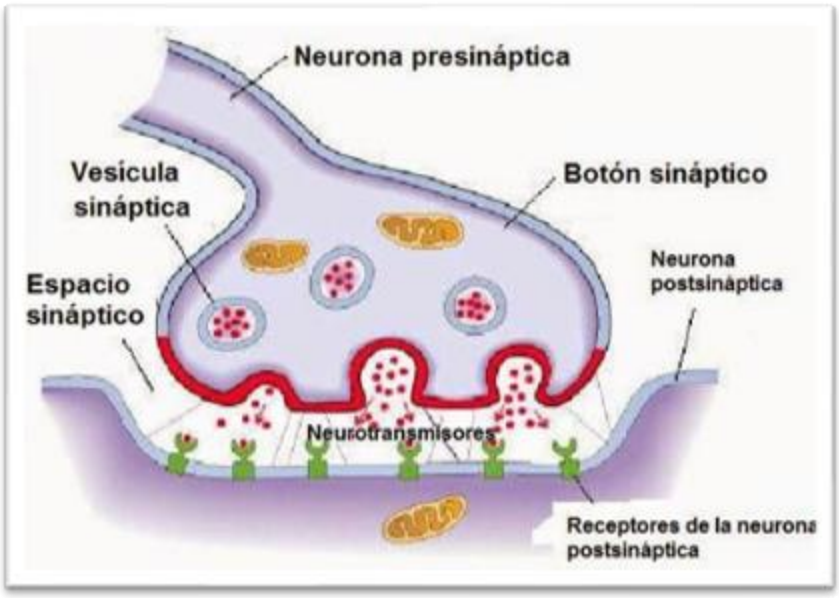


Ilustración 4. Diagrama básico de la sinapsis química

Para realizar el proceso de aprendizaje, en el botón sináptico se encuentran una serie de neurotransmisores de los cuales se modifica su cantidad para así poder variar el potencial del pulso creado en la dendrita de la neurona receptora. De esta forma se genera una respuesta diferente según el sumatorio de señales acumulado en la neurona receptora, por lo que el aprendizaje de una red neuronal biológica se basa en la modificación de la carga de las conexiones entre las diferentes neuronas para poder alcanzar la respuesta correcta.

vi. Redes neuronales artificiales

Las redes neuronales artificiales [4] forman parte del campo del aprendizaje supervisado y emulan el funcionamiento de una red neuronal bilógica, en donde a partir de unos datos de entrada se obtiene una respuesta como salida.

Para realizar este proceso se requiere de una unidad básica que en este caso es el perceptrón y a través de la combinación de estas unidades se puede crear una red. Cada uno de los perceptrones reproduce una función en base a las entradas recogidas, la cual emitirá la señal de salida que se enviará a otra unidad.

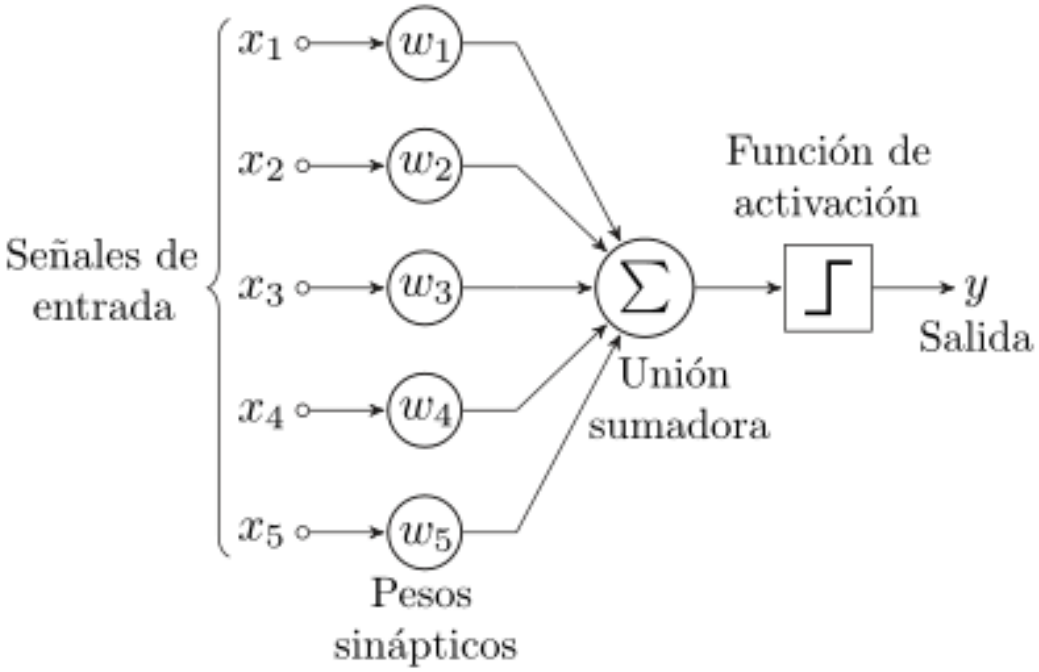


Ilustración 5. Esquema de una neurona artificial

El proceso de aprendizaje en este tipo de redes neuronales consiste en ajustar los pesos en los perceptrones para conseguir la salida que se quiere a partir de las entradas. Por ello, la salida de cada uno de estos será la activación del sumatorio multiplicado por el peso sináptico asociado.

La función que se aplica sobre el sumatorio se denomina función de activación y su trabajo consiste en calcular la salida en función de las entradas y pesos. Así como en una red neuronal biológica se utiliza un umbral para el potencial de acción, en una red neuronal artificial se pueden aplicar diferentes funciones de activación. En este caso, solo se va a introducir la función de activación denominada ReLu y esta se define como:

$$\mathbf{ReLU\,(x) = max(0, x)}$$

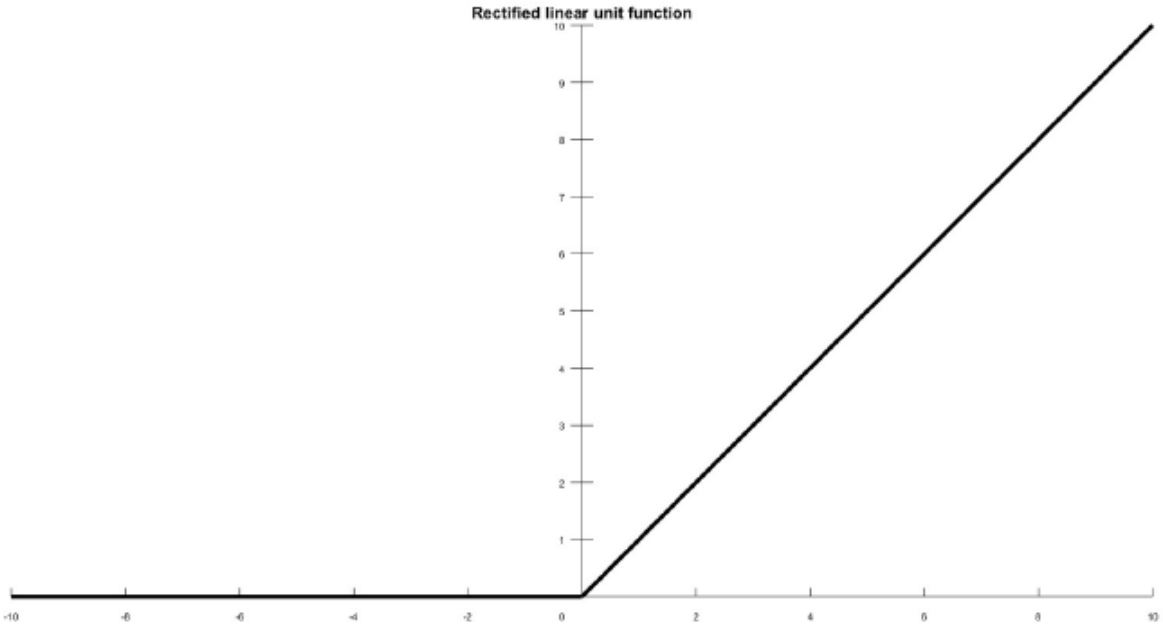


Ilustración 6. Representación de la función de activación ReLu.

Por otra parte, un apartado importante dentro de la red neuronal artificial es la creación de la arquitectura de esta. Muchos factores intervienen en la complejidad de su forma y no existe una norma escrita que permita saber cuál es la arquitectura óptima en cada problema, por lo que la consecución de la opción óptima se alcanza a través del proceso de prueba y error.

Como se muestra en la figura, las redes se forman a partir de tres tipos de capas según su localización:

- Capa de entrada: recibe los datos, tan solo es una capa y tendrá tantos nodos como datos de entrada se introduzcan.
- Capa oculta: capas que se localizan entre la capa de entrada y la de salida que se encargan de procesar los datos para alcanzar los resultados.
- Capa de salida: su trabajo es sacar los datos que forman parte de la respuesta y también está formada solamente por una capa como la capa de entrada.

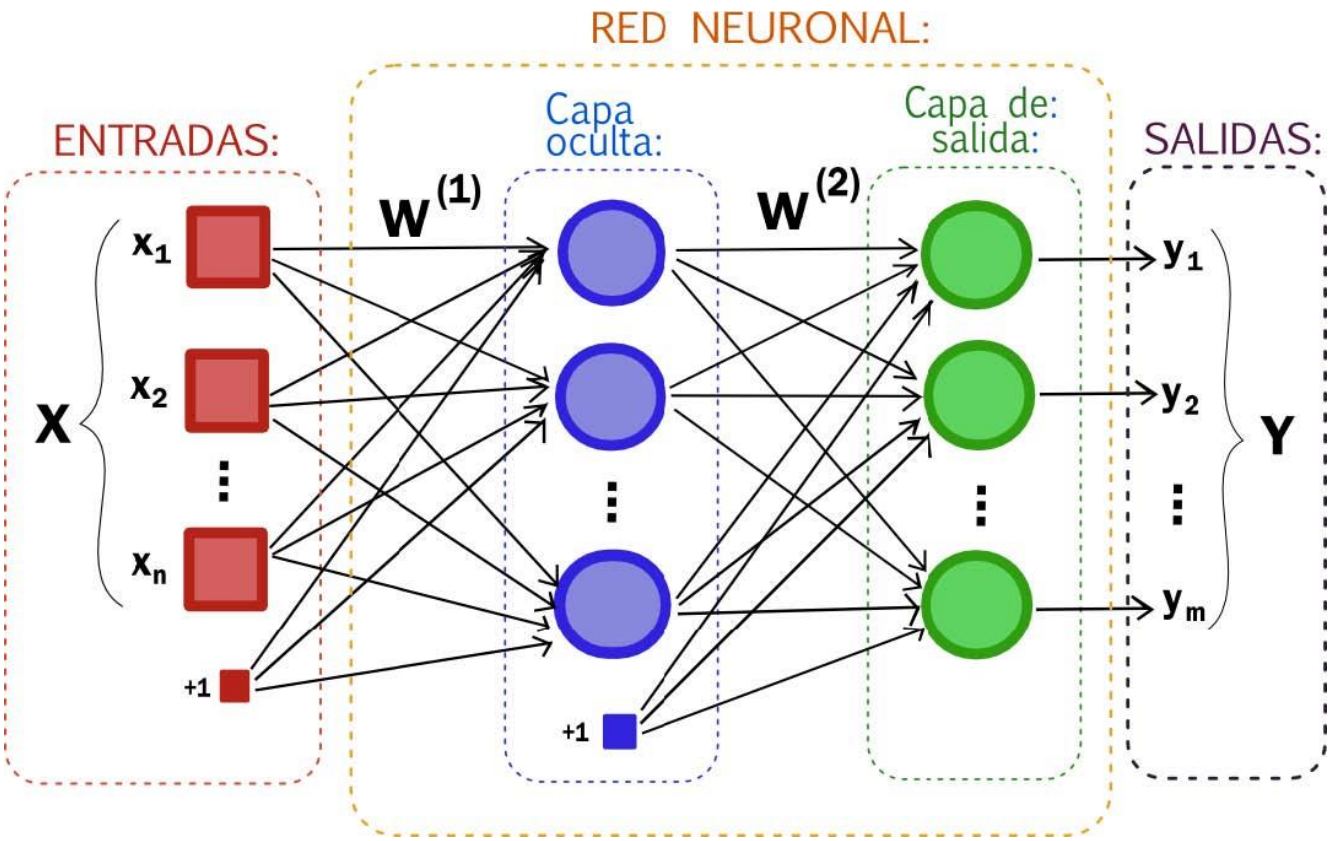


Ilustración 7. Esquema de ejemplo de una arquitectura de red neuronal.

vii. Redes neuronales convolucionales

Las redes neuronales convolucionales [5] son un tipo de red neuronal artificial que procesan sus capas imitando el comportamiento del ojo humano con el objetivo de identificar diferentes características en las entradas. Para cumplir con ese objetivo, este tipo de redes contienen varias capas ocultas especializadas. Estas capas son las siguientes:

- **Capas Convolucionales**: el objetivo principal de estas capas consiste en el proceso de realización de una convolución sobre la entrada en sus dimensiones, donde en la salida de un elemento en concreto participarán el y los vecinos de su alrededor. Hay que añadir que el filtro que se utiliza estará formado las mismas dimensiones que tenga la entrada. Por lo tanto, la salida de la capa convolucional estará formada por el número de dimensiones que tenga la entrada y el será la convolución del *kernel* y los datos pertenecientes a la entrada tras la aplicación de la función de activación en cada uno de los elementos.
- **Capas de submuestreo**: la meta de este tipo de capa es reducir el número de parámetros calculados y el tamaño de la imagen. El tipo de submuestreo que se utiliza comúnmente es el submuestreo por máximos (*maxpooling*), donde se coge el máximo valor dentro de un grupo de píxeles y se descartan los demás. El uso de la capa de submuestreo por máximos permite alcanzar en menor tiempo la convergencia y provoca una disminución de los datos a calcular.
- **Capa completamente conectada**: esta capa se localiza en el final de la arquitectura de la red convolucional (pueden ser más de una capa) y su objetivo es transformar la dimensionalidad del problema que se está abordando. Recibe las imágenes generadas tras el proceso que se ha ido realizando en las demás capas como entrada y en según el valor que ofrecen los píxeles, asigna un valor numérico por imagen, representando así, la probabilidad de pertenencia a una clase.

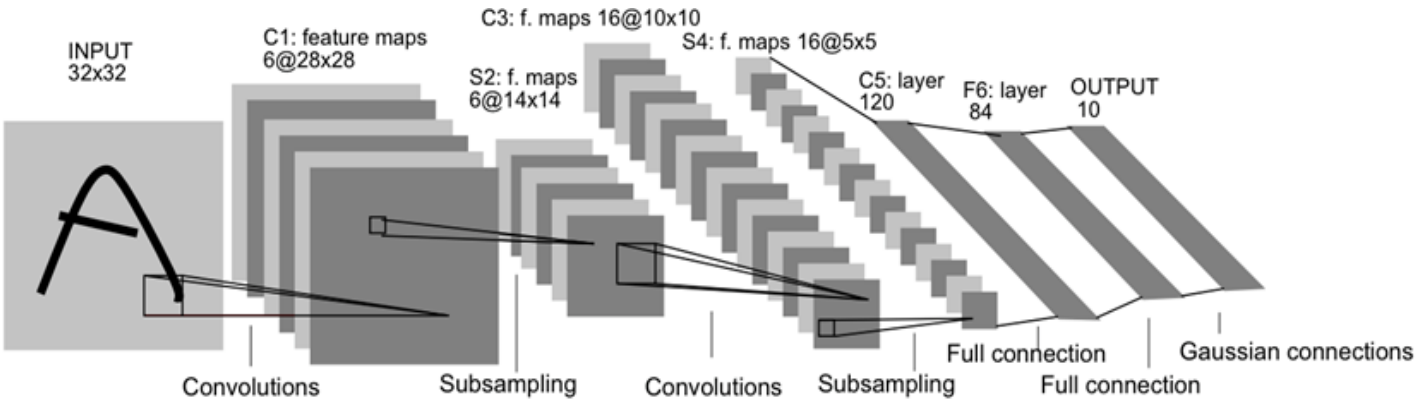


Ilustración 8. Esquema genérico de una red neuronal convolucional.

La arquitectura de las redes neuronales convolucionales ha quedado explicada de una forma reducida, pero la operación de convolución que forma la base del algoritmo todavía no se ha definido y es un elemento diferenciador.

Esta operación de convolución (definida en la figura 9999) tiene trata de superponer un filtro sobre una imagen e ir deslizando este sobre la propia imagen hasta haber cubierto todas las regiones disponibles. En cada paso del deslizamiento del filtro, se calcular el valor que resulta de la suma de todas las multiplicaciones de cada píxel por el valor que corresponda a la posición del filtro y con este procedimiento se genera la matriz resultado.

Una de las características para tener en cuenta es la modificación del *padding*. En algunas ocasiones es recomendable rellenar con ceros los límites de la imagen, ya que permite mantener bajo control la dimensionalidad de las salidas. El proceso se consigue con una adición de ceros entorno a los bordes de la entrada modificando a su vez, el número de filas y columnas.

Este tipo de redes ofrecen una gran eficiencia computacional. El algoritmo puede utilizar equivalencia de pesos en diferentes secciones de la imagen, reduciendo la exigencia de cómputo. Además, los valores de salida tan solo dependen de un grupo reducido de valores en la entrada y no forman una conexión directa, lo que conlleva a un rendimiento mayor.

viii. Entrenamiento

Conseguir minimizar la función de error en Deep Learning con el conjunto de entrenamiento no es una tarea trivial. La cantidad de datos necesaria en el algoritmo junto con la configuración de los parámetros de este es la principal causa del problema. En esta sección se tratan varios aspectos que componen el entrenamiento:

- **Inicialización de pesos**: en el Deep Learning la tendencia de los gradientes tiene dos tipos de comportamiento: hacerse cada vez más pequeños o grandes. Esto implica que surjan problemas en la convergencia.

  Una solución para la selección de pesos a la hora de inicializar es escoger los pesos según el tipo de función de activación que emplee la red neuronal.
- **Regularización**: el objetivo de la regularización es realizar una modificación en el modelo para evitar que este se ajuste en exceso al conjunto de entrenamiento. Existen diferentes tipos de regularización y según el tipo de red que se vaya a trabajar conviene seleccionar uno de ellos.
- **Batch Size**: la gran cantidad de datos que se tendrían que lanzar a la red neuronal para poder entrenarla es inviable. Por este motivo, se dividen los datos de entrenamiento en conjuntos de datos de menor tamaño.

  Al dividir en lotes los datos conseguimos algunas ventajas interesantes. Por ejemplo, se requiere menos memoria, ya que, al entrenar la red con menos muestras, el procedimiento de entrenamiento general requiere una cantidad menor de memoria. Esto conlleva a un entrenamiento en menor tiempo, porque se actualizan los pesos justo después de cada propagación.

  El valor del tamaño de estos lotes es un hiperparámetro dentro del modelo y según el conjunto de entrenamiento que se maneje hay que configurarlo para que los lotes entren en la memoria sin problema.

- **Batch Normalization**: es una técnica para acelerar el proceso de aprendizaje. El objetivo del Batch Normalization es normalizar las entradas a cada capa de la red por cada uno de los lotes. De esta forma se reduce el número de propagaciones requeridas en el entrenamiento.
- **Ratio de aprendizaje**: dentro de los métodos de optimización en Deep Learning es uno de los aspectos más destacados. Es el hiperparámetro que marca la velocidad con la que se avanza en el proceso de aprendizaje y se define como $\boldsymbol{\alpha}$.

  Al tratarse de un hiperparámetro, la elección del mejor valor para este se consigue realizando una serie de configuraciones y viendo cual es el que mejor resultado arroja. Si el valor de $\boldsymbol{\alpha}$ es demasiado grande, será difícil encontrar los coeficientes que minimicen la función de coste, y si el valor de $\boldsymbol{\alpha}$ es demasiado pequeño, el gradiente descendiente tardará mucho en encontrar la solución adecuada.

  El objetivo principal del ratio de aprendizaje es alcanzar una convergencia en el menor tiempo posible. Pero según el problema que se enfrente, el valor de $\boldsymbol{\alpha}$ variará, por lo que existen técnicas que proponen que este no sea prefijado y que decrezca en el proceso de consecución del mínimo.
- **Optimización:** existen diferentes métodos de optimización utilizados en Deep Learning y uno de ellos es RMSprop.

  RMSprop es una extensión del descenso por gradiente y la versión AdaGrad [6]. A través del uso una media móvil decreciente permite que el algoritmo no tenga en cuenta los gradientes tempranos y se centre en los gradientes observados recientemente durante el proceso de búsqueda.

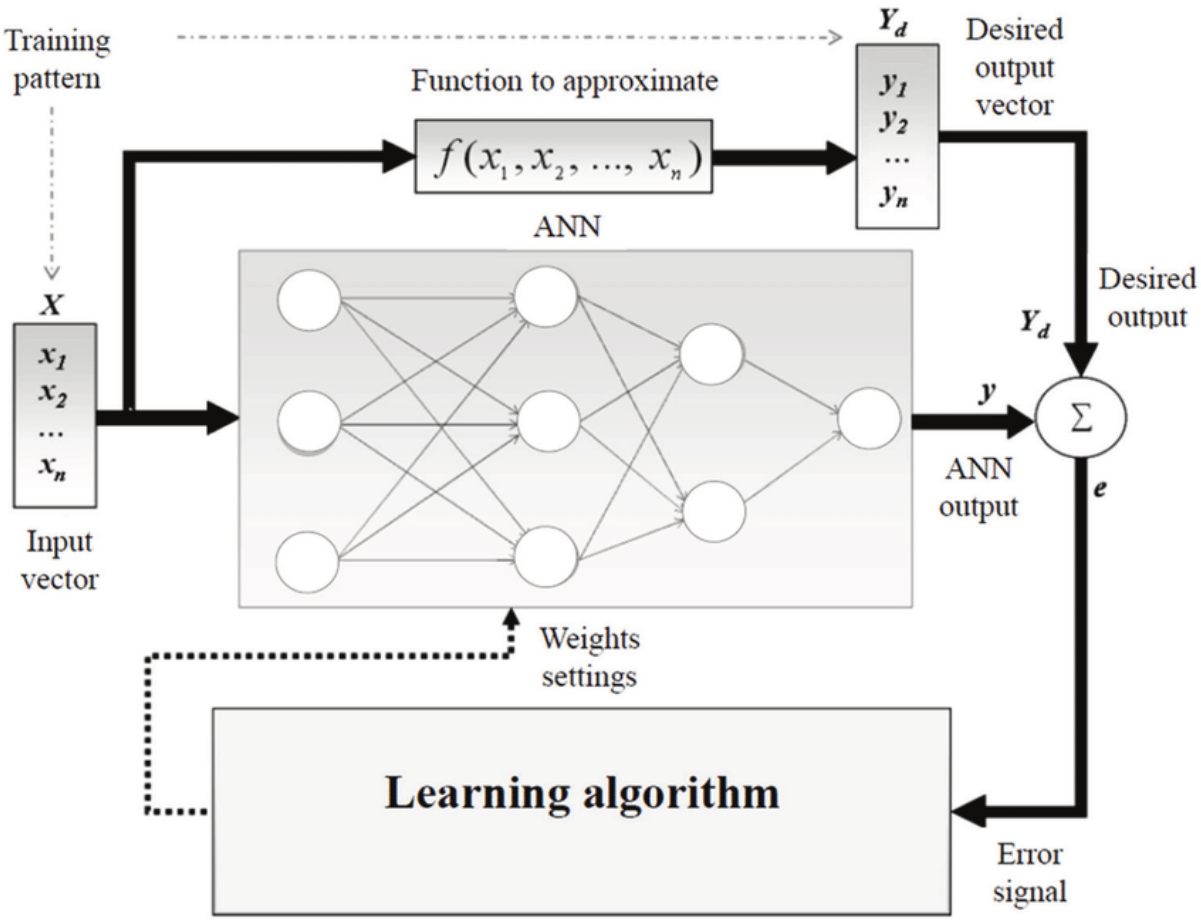


Ilustración 9. Diagrama representativo de entrenamiento de una red neuronal artificial.

c. DQN

En el presente apartado se procede a desarrollar el concepto teórico de DQN (Deep Q Network), el cual es una combinación de redes neuronales profundas con el algoritmo de aprendizaje por refuerzo Q-learning. Este último funciona bien cuando el entorno que se va a tratar es simple y la función Q(s,a) queda representada como una matriz de valores. El problema aparece cuando se establecen una gran cantidad de estados y acciones diferentes, ya que la matriz del algoritmo se expande hasta alcanzar un tamaño muy grande y pasa a ser no viable la utilización de esta.

Para la resolución de este problema, se creó el algoritmo DQN [7], del que han surgido otros algoritmos que son una evolución de este.

La peculiaridad que diferencia al algoritmo Q-Learning es la técnica que realiza para calcular los valores Q (o estimarlos). En el caso de DQN, se utilizan dos redes neuronales con el pretexto de estabilizar el proceso de aprendizaje [8]. Por un lado, la primera red neuronal principal, está representada por los parámetros θ. Esta red se utiliza para estimar los valores Q del estado s y la acción a del presente: Q (s, a; θ). La segunda, la red neuronal objetivo, parametrizada por θ´, tendrá la misma arquitectura que la red principal, pero se utilizará para aproximar los valores-Q del siguiente estado **s´** y la siguiente acción **a´**.

El aprendizaje se produce en la red principal y no en la red objetivo. La red objetivo se bloquea, es decir, sus parámetros no se cambian durante varias iteraciones y después los parámetros de la red principal se copian a la red objetivo, transmitiendo así el aprendizaje de una a otra, haciendo que las estimaciones calculadas por la red objetivo sean más precisas.

$$Q(s,a;\theta) = r + \gamma \max_{a'} Q(s',a';\theta')$$

Ilustración 10. Ecuación de Bellman en DQN.

Para poder entrenar la red neuronal en cuestión, se necesita una función de pérdida o coste, que se define como el cuadrado de la diferencia entre ambas partes de la ecuación de Bellman:

$$L(\theta) = \mathbb{E}[(r + \gamma \max_{a'} Q(s', a'; \theta') - Q(s, a; \theta))^2]$$

Ilustración 11. Función de pérdida para entrenar el algoritmo DQN.

Por lo tanto, esta será la función que se minimice utilizando el algoritmo de descenso de gradiente.

Cabe añadir que en este algoritmo la técnica de exploración utilizada se denomina Epsilon-Greedy. En la técnica mencionada se define el término ϵ, que se utiliza en el entrenamiento y cuando se selecciona la acción a tomar, se escoge la acción indicada por la política del agente con probabilidad 1- ϵ o en el caso de una acción uniformemente aleatoria, con ϵ.

3. Herramientas

En el presente capítulo se presentan las diferentes herramientas utilizadas en el desarrollo del proyecto. A lo largo de este se presentarán una a una cada una de ella junto con los aspectos más importantes relacionados con la implementación.

a. Python

El lenguaje de programación utilizado para el desarrollo del proyecto ha sido el lenguaje de propósito general Python en su versión 3.8.8.

En los últimos años Python se ha posicionado como el lenguaje de programación más utilizado en el campo de la inteligencia artificial, con una serie de características que favorecen el desarrollo de proyectos de IA:

- **Compilación:** Python se considera un lenguaje completo. El desarrollador puede hacer uso de forma directa ya que no necesita ser compilado.
- **Legibilidad:** es un lenguaje diseñado desde un inicio enfocado en la buena legibilidad del código. Los algoritmos de IA tienden a ser complejos, por lo que una buena legibilidad favorece al desarrollo de estos.
- **Comunidad:** actualmente es uno de los lenguajes más populares del mundo y en el campo de la IA ha escalado puestos hasta colocarse en lo más alto. Todo ello ha provocado un aumento de la comunidad, lo que favorece ha una especialización en el campo (más librerías, herramientas de software, código abierto, etc.)
- **Rendimiento**: A pesar de no ser código compilado y ser un lenguaje de alto nivel, las implementaciones de Python más utilizadas habitualmente compilan a código intermedio.
- **Gestión de memoria**: tiene gestión de memoria automática, por lo que acepta una variedad de paradigmas de programación.

- **Multiplataforma:** al ser un lenguaje tan extendido se puede desarrollar en todas las plataformas principales.
- **Librerías**: son una de las características principales. La popularidad del lenguaje ha permitido el desarrollo de librerías estables y con una extensa documentación que facilitan el desarrollo en gran medida. Para el desarrollo de IA esto es importante y destacan las siguientes:
    - Numpy: cálculo vectorial, matricial y tensorial.
    - SciPy: colección de herramientas y algoritmos matemáticos.
    - Scikit-Lear: diferentes algoritmos de aprendizaje automático.
    - Pytorch: marco de aprendizaje automático.
    - Matplotlib: Para representación gráfica.

ix. Alternativas a Python en inteligencia artificial

Actualmente, existen otros lenguajes de programación que pueden ser utilizados en este campo. Por una parte, R, un lenguaje de programación orientado al análisis estadístico que dispone de librerías dedicadas al aprendizaje automático. El problema de este lenguaje en el desarrollo de IA es que, debido a la especialización en el análisis estadístico que posee, es más complejo.

Por otra parte, existe C++, que es un lenguaje de propósito general orientado a objetos como Python, pero se localiza en otro nivel más bajo. Es un lenguaje más utilizado en el campo de la robótica, sensores, etc. Se aleja un poco de la línea de este proyecto y al igual que R, es más complejo que Python en este caso, por lo que el seleccionado para el proyecto (Python) es el lenguaje correcto.

b. Anaconda

Anaconda es una suite de código abierto que abarca una serie de aplicaciones, librerías y conceptos para el desarrollo de la ciencia de datos con Python [9].

En líneas generales Anaconda Distribution es una distribución de Python que funciona como un gestor de entorno, un gestor de paquete y que posee una amplica colección de paquetes de código abierto.

La adopción de esta herramienta de desarrollo es causa de las características que dispone:

- **Comunidad:** cuenta con una comunidad extensa y una documentación muy detallada con código abierto.
- **Interfaz gráfica**: es sencilla e intuitiva.
- **Paquetes y dependencias**: permite instalar paquetes y dependencias en entornos para la ciencia de datos personalizados por el usuario.
- **Ejecución**: permite compilar Python en código de máquina para una ejecución rápida y facilita la escritura de complejos algoritmos paralelos para la ejecución de tareas.
- **Soporte**: cuenta con soporte para la computación de alto rendimiento.
- **IDE`s:** ayuda a desarrollar proyectos de ciencia de datos utilizando diversos IDE, en este caso, Jupyter.

c. Pytorch

PyTorch es un paquete de Python diseñado para realizar cálculos numéricos haciendo uso de la programación de tensores [10]. Por otra parte, también ofrece la opción de ejecución en GPU para acelerar los cálculos.

Este paquete es usado para sustituir a numpy y procesar los cálculos en GPU y está centrado principalmente en el desarrollo de redes neuronales.

Se ha decidido utilizar PyTorch porque dispone de tutoriales y manuales actualizados y la comunidad está activa actualmente.

Dispone de una interfaz sencilla para la construcción de redes neuronales sin necesidad de una librería a nivel superior como Tensorflow, Keras, etc. También trabaja con grafos dinámicos, lo que conlleva que en tiempo de ejecución se puedan ir modificando funciones (cálculo del gradiente varía a su vez).

Por otra parte, PyTorch dispone de soporte para ejecución en tarjetas gráficas utilizando internamente CUDA, la API que conecta la CPU con la GPU perteneciente a NVIDIA.

x. Instalación:

La instalación del paquete PyTorch para la utilización de la GPU Nvidia en los cálculos con tensores (PyTorch dispone de soporte para su ejecución en tarjetas gráficas (GPU), utiliza internamente CUDA, una API que conecta la CPU con la GPU que ha sido desarrollado NVIDIA) no es trivial, por lo que se ofrece una visión de la instalación:

Descargar e instalar Anaconda

1.1 Ir al sitio oficial de Anaconda y descargar el instalador adecuado a la configuración del equipo.

1.2 En el apartado de la configuración seleccionar la instalación de la versión Python conveniente.

Instalar PyTorch

1.3 Utilizar las instrucciones oficiales de instalación local de PyTorch.

1.4 Elegir la configuración deseada, en este caso, se selecciona el paquete CUDA en las opciones (es necesario que el sistema admita GPU Nvidia).

Ilustración 12. Selección de configuración para la instalación de PyTorch.

1.5 Ejecutar en terminal el comando generado en la web debajo del panel de opciones de la configuración.

Verificar la instalación

Para verificar que la instalación ha concluido con éxito, hay que abrir una terminal e ingresar "python" para iniciar el intérprete de Python y escribir: "import torch".

Si el controlador de GPU y CUDA están habilitados se verifica con la respuesta del siguiente comando: ***"nvcc –version"***

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2021 NVIDIA Corporation
Built on Wed_Jun__2_19:25:35_Pacific_Daylight_Time_2021
Cuda compilation tools, release 11.4, V11.4.48
Build cuda_11.4.r11.4/compiler.30033411_0
```

Ilustración 13. Comprobación del controlador GPU Y CUDA.

```
one_d = torch.arange(0,16)
print(one_d)
two_d = one_d.view(4,4)
print(two_d)
print(two_d.size())
```

```
tensor([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
tensor([[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11],
        [12, 13, 14, 15]])
torch.Size([4, 4])
```

Ilustración 14. Ejemplo de PyTorch operativo.

d. CUDA

Es una plataforma de computación paralela y un modelo de programación que mejora significativamente el rendimiento de la computación al utilizar las funciones de la unidad de procesamiento de gráficos (GPU) [11].

Esta plataforma provee un conjunto de extensiones para lenguajes de programación para poder admitir la implementación directa de algoritmos paralelos. Con CUDA, los programadores pueden enfocarse en la paralelización de algoritmos en lugar de perder tiempo implementando algoritmos.

Por otra parte, CUDA soporta computación heterogénea usando CPU y GPU al mismo tiempo para aplicaciones. La parte serial de la aplicación se ejecuta en la CPU y la parte paralela se ejecuta en la GPU. La CPU y la GPU se consideran dispositivos independientes con su propio espacio de memoria, esta configuración también permite cálculos simultáneos en la CPU y la GPU sin competir por los recursos de memoria.

Las GPU que aceptan CUDA tienen cientos de núcleos y son capaces de ejecutar miles de subprocesos informáticos juntos. La memoria compartida en el chip permite que las tareas paralelas que se ejecutan en estos núcleos compartan datos sin enviar datos a través del bus de memoria del sistema.

e. OpenAI Gym

OpenAI Gym es un conjunto de herramientas para desarrollar y comparar algoritmos de aprendizaje por refuerzo. Está formado por una seriede entornos de problema que se pueden utilizar de manera sencilla para la investigación y análisis de los algoritmos.

Los entornos que lo forman se definen a través de una interfaz que tiene como objetivo la definición de estos. Esto permite mudar de un entorno a otro fácilmente, lo que brinda la oportunidad de hacer más reproducibles los resultados de los algoritmos en los experimentos.

Se puede señalar también que OpenAI Gym ofrece la oportunidad de compartir y reproducir los resultados de los algoritmos creados, para su comparación, mediante su página web. En esta aparece un listado de los usuarios con las puntuaciones obtenidas en sus algoritmos ordenados cronológicamente.

Hoy en día, OpenAI Gym comparte un registro amplio de entornos que varían según sus características y dentro de los tipos existen experimentos que tienen diferentes rangos de dificultad. Los entornos disponibles son:

- **Toy Text:** entornos de texto simples para comenzar.
- **Algorithms**: aprender a imitar cálculos.
- **Atari:** juegos 2D de la consola Atari, que han sido históricamente utilizados para la investigación de algoritmos de aprendizaje por refuerzo.
- **Box2D:** sistemas de control en 2D. El motor de física y las acciones son más compleja.
- **Classic Control:** sistemas de control sencillos. Con motores de física simple y acciones a realizar de número reducido.
- **Robotics:** tareas simuladas basadas en objetivos para los robots Fetch y ShadowHand.
- **MuJoCo:** tareas de control continuo, que se ejecutan en un simulador de física rápido.
- **Proyectos de terceros:** entornos creados por la comunidad.

En el trabajo de investigación desarrollado se han escogido dos entornos de los disponibles en la página web y se ha hecho un ejercicio de creación y prueba de algoritmos basados en DQN:

- **CartPole**: perteneciente a el apartado de entornos "Classic Control"; en este se controla un carro que puede moverse unidimensionalmente. Cuenta con un poste vertical que puede girar libremente.

  El objetivo es que el carro aprenda de forma autónoma a mantener el poste en vertical y evitar que se caiga durante el mayor tiempo posible.

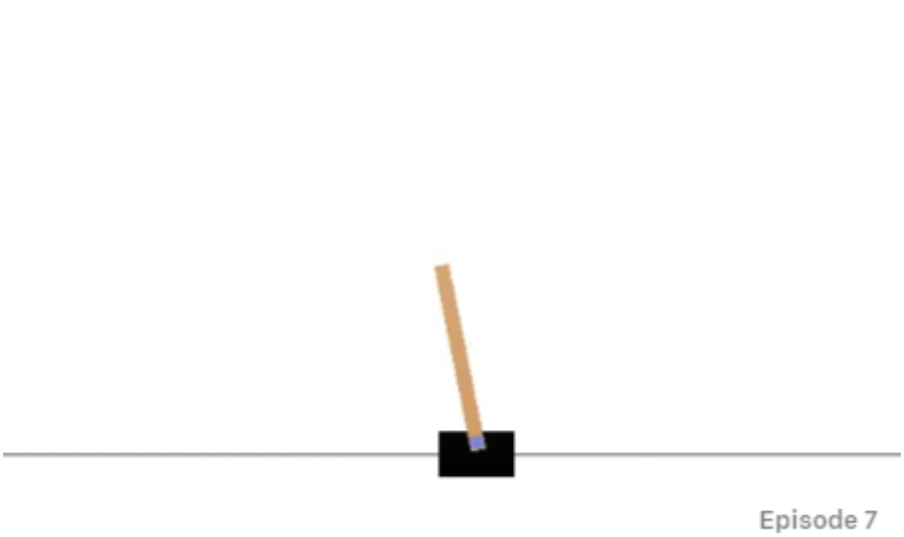


Ilustración 15. Ejemplo fotograma en CartPole-v0.

- **CarRacing**: perteneciente al apartado de entornos "Box2D"; trata sobre un entorno de carreras. El estado consta de 96x96 píxeles con un entorno RGB clásico. La recompensa es igual a -0,1 por cada cuadro y + 1000 / N por cada mosaico de pista visitado, donde N está representado por el número total de mosaicos a lo largo de la totalidad de la pista. Además, hay una barrera fuera de la pista, lo que resulta en una penalización de -100 y un final inmediato del episodio si se cruza.

El exterior de la pista está formado por césped, que no da recompensa pero que, debido al roce del entorno, da como resultado una lucha para que el vehículo vuelva a entrar en la pista.

En la renderización se muestran en la parte inferior de la ventana varios indicadores que muestran la información de los sensores y el estado del búfer de RGB. De izquierda a derecha: velocidad real, cuatro sensores ABS, posición del volante, giroscopio.

El objetivo es que el agente aprenda de forma autónoma a seguir el recorrido de la carretera sin salirse.

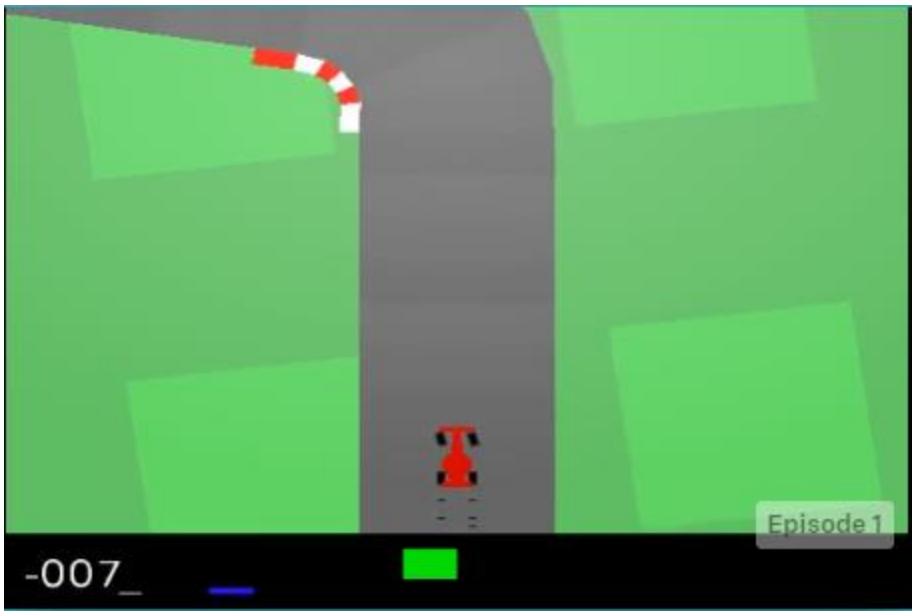


Ilustración 16. Ejemplo fotograma en CarRacing-v0.

4. Implementación y despliegue

Este capítulo presenta las partes más relevantes de la implementación, enfocándose especialmente en el desarrollo relacionado con los objetivos del trabajo de investigación.

a. Algoritmos

A lo largo del proyecto se han implementado dos algoritmos relacionados entre ellos, al ser uno, una evolución del otro. Los algoritmos de Aprendizaje por Refuerzo Profundo utilizados han sido DQN y DDQN (Double DQN).

Las secciones posteriores presentan las características de los algoritmos implementados y se ofrece un esquema general de la base en la implementación de los algoritmos [12]:

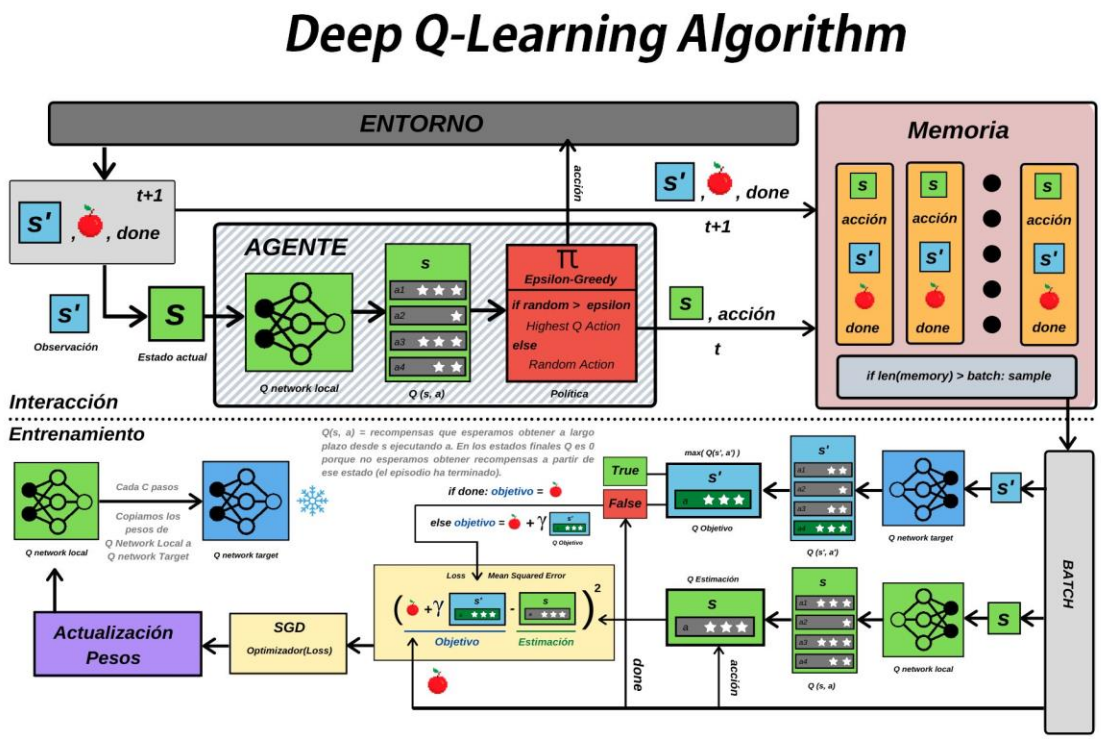


Ilustración 17. Diagrama de Deep Q-Learning - Implementación del proyecto.

xi. DQN

Como ya se mencionó, el algoritmo DQN se basa en el uso de redes neuronales utilizadas para estimar el valor Q, llamadas redes Q. Para lograrlo, se utiliza la biblioteca PyTorch, que ayuda en gran medida en el proceso de creación de redes neuronales. La segunda parte del algoritmo DQN es el método iterativo de aprendizaje Q, un método para calcular el error entre el valor Q actual estimado y el nuevo valor Q calculado. El error mencionado se trata de minimizar con el paso iterativo, para así llegar a mejores estimaciones de los valores de Q. PyTorch por su parte, también ofrece diferentes algoritmos de optimización que han sido implementados en el desarrollo para poder minimizar el error.

xii. DDQN

El problema con el algoritmo DQN es que sobreestima las recompensas reales; es decir, los valores-Q que aprende piensan que van a obtener una recompensa mayor de lo que en realidad obtendrá. Para solucionarlo, los autores del algoritmo Double DQN [13], proponen una solución: separar la selección y evaluación de la acción en dos pasos diferentes.

Para conseguir la implementación de la DDQN, se cambia la ecuación de Bellman del algoritmo DQN y se convierte en:

$$Q(s,a;\theta) = r + \gamma Q(s', argmax_{a'} Q(s',a';\theta);\theta')$$

Ilustración 18. Modificación de la ecuación de Bellman en el algoritmo DQN.

Primero la red neuronal principal θ decide cuál es la mejor acción entre todas las posibles, y luego la red objetivo evalúa esa acción para conocer su valor-Q. Este simple cambio ha demostrado reducir las sobreestimaciones y resultar en mejores políticas.

b. Secciones

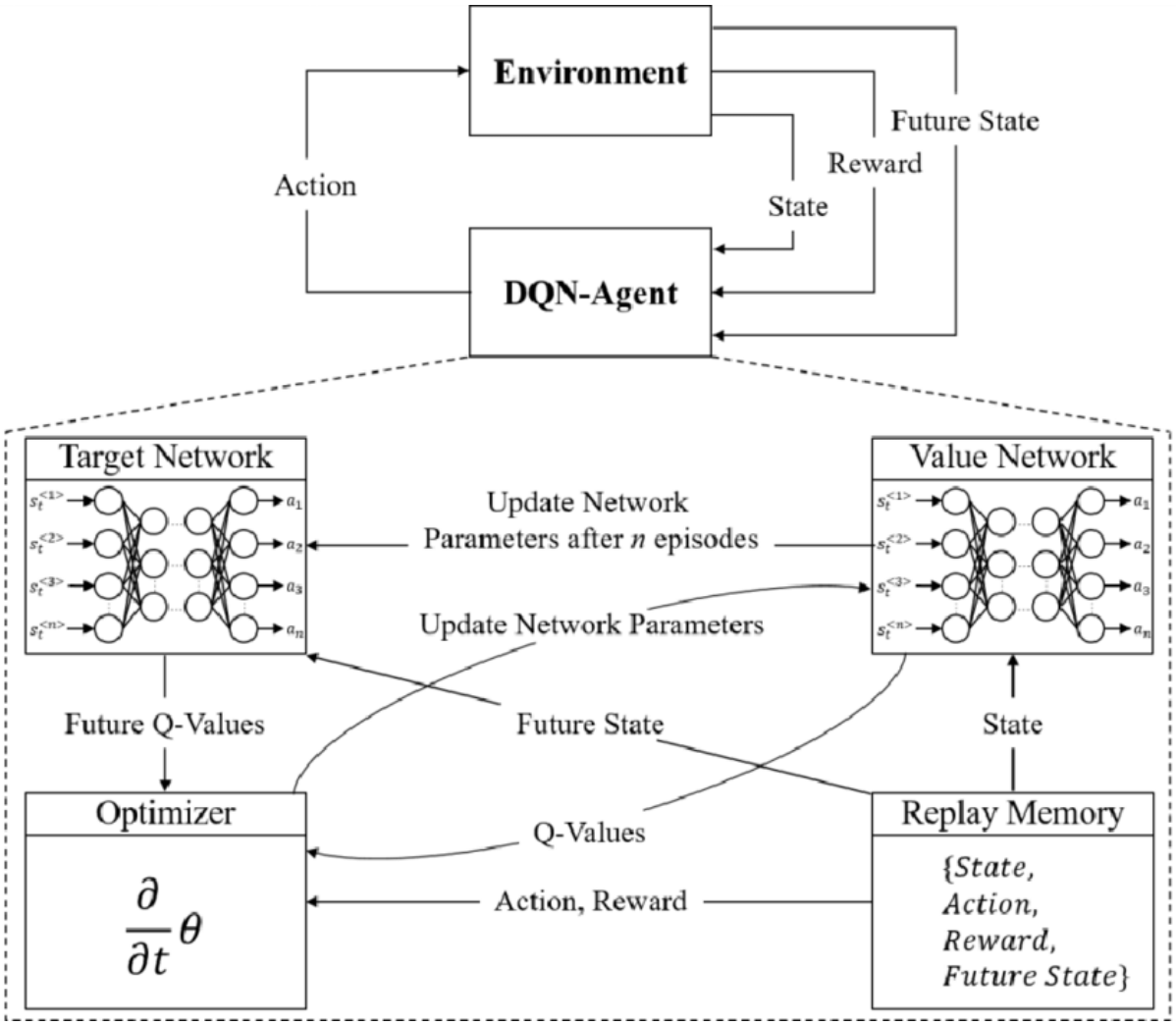


Ilustración 19. Flujo generado en el algoritmo DQN en las redes.

La construcción para la implementación de los algoritmos consta de varias secciones fijas de código que combinadas reproducen el resultado. Estas secciones son las siguientes:

xiii. Paquetes

Es la primera sección y en ella se importan todos los paquetes necesarios para la implementación. Los paquetes utilizados son variados y con diversas funciones, como por ejemplo el paquete de PyTorch para la construcción de las redes neuronales. También, en esta

sección se importa el entorno originario de OpenAI Gym junto con la asignación de CUDA para los cálculos posteriores de la GPU.

xiv. Replay Memory

En esta sección se construye a través de una clase la repetición de experiencia. Con esta se consigue la reproducción de experiencias, donde se almacenan las experiencias del agente en cada paso de tiempo en un conjunto de datos.

En el algoritmo DQN, la repetición de experiencia se ha transformado de manera que se priorice su manejo. Las transiciones para utilizar a lo largo del entrenamiento antes del cambio se seleccionaban de forma aleatoria según se generaban por parte del agente, pero se ha creado un cambio donde se priorizan en más instantes las transiciones de las que se ha aprendido un resultado relacionado con el objetivo y así, el entrenamiento mejora y pasa a ser más efectivo.

xv. Q Network

Esta sección ocupa la parte de la creación del modelo a través de una red convolucional que toma la diferencia entre los cuadros de pantalla actuales y los anteriores. Su objetivo es tratar de predecir el retorno esperado de tomar cada acción dada la entrada actual.

En esta parte de la implementación se almacenan todas aquellas capas donde se tienen parámetros que van a ser entrenador por el sistema.

La red convolucional que se utiliza se utilizan diferentes tipos de capas:

- Capas convolucionales: contienen los filtros, kernel y vas analizando las escenas

- Capas de “batch normalization”: normalización de los conjuntos de ejemplos.

- Capas de activación: la función de activación utilizada en este proyecto es la función ReLu.

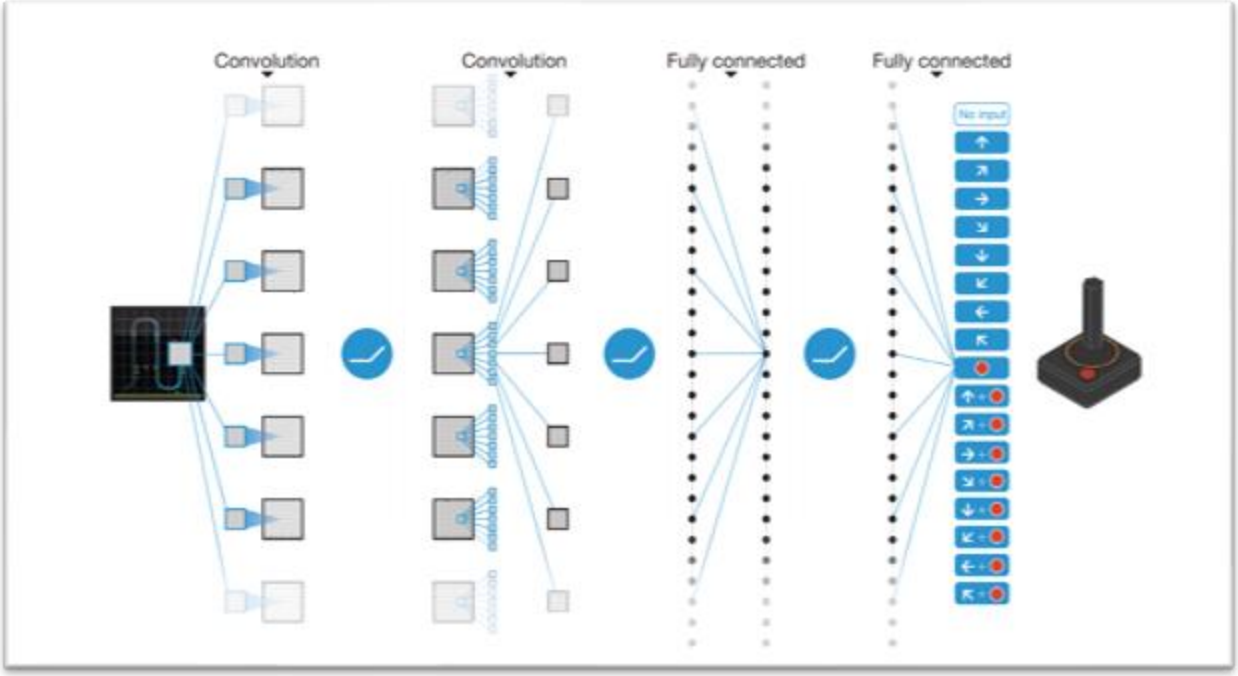


Ilustración 20. Arquitectura de la red convolucional en el algoritmo DQN.

xvi. Extracción de los movimientos

El código referente a esta parte se utiliza para extraer y procesar imágenes renderizadas del entorno. Utiliza el paquete de "torchvision", que facilita la composición de transformación de imágenes.

Se puede ver como la infraestructura que rodea a la red neuronal. Por una parte, crea una lista de los diferentes procesamientos que se quieren hacer con cada una de las imágenes, en este caso, se convierten los tensores a imágenes de la clase PIL que es la que tiene las funcionalidades interesantes para hacer las transformaciones. Por otra parte, existen unas funciones para recoger los datos de la pantalla y los estados del agente con los sensores que se pueden medir del sistema del agente.

xvii. Entrenamiento de la red

En el entrenamiento de la red se crea una instancia del modelo y su optimizador. En esta sección se crean primero una seria de parámetros fijos para después poder entrenar las redes. Estos parámetros fijos son los siguientes:

- **Batch size**: este es el número de ejemplos que se introducen en la red para que entrene de cada vez. Si el número es pequeño, significa que la red tiene en memoria poca cantidad de datos, y entrena más rápido.
- **Gamma**: factor de descuento que se utiliza para penalizar a las acciones futuras que tomemos para tenerlas menos en cuenta
- **Valor de épsilon al inicio**
- **Valor de épsilon al final**
- **Decaimiento de épsilon**
- **Target update**: variable utilizada para marcar cuando se realiza la actualización de la red objetivo, copiando todos los pesos y bias en DQN.

Seguidamente se obtiene el tamaño de la pantalla para poder inicializar las capas correctamente según la forma del gimnasio de OpenAI gym y se construyen la "policy net" y la "target net" junto con la llamada al optimizador.

El núcleo de la sección es la función "**select_action()**", que seleccionará una acción de acuerdo con una política de épsilon.. La probabilidad de elegir una acción aleatoria comenzará en el valor de épsilon inicial y disminuirá exponencialmente hacia el valor de épsilon inicial. La última variable relacionada a épsilon controlará la tasa de decaimiento (aleatoriedad modulada con un epsilon inicial - final - decaimiento).

xviii. Optimización del modelo

En esta sección se crea la lógica que define toda la función del algoritmo. Se realiza una optimización del modelo donde cada una de las transiciones se concatenan y se genera una nueva dimensión que va a ser la del "batch size".

Dentro de la función se realizan varios cálculos para llegar al objetivo. El primer cálculo es el cálculo de una máscara de estados no finales que se concatena con los elementos del lote. Seguidamente se pasa a la generación de los tensores necesarios (estado, acción, recompensa) y se calculan los valores esperados de Q. Una vez tenemos estos valores se utiliza la función de error de Huber para obtener la pérdida y se optimiza el modelo con entrenamiento iterativo en la "policy net".

xix. Ejecución

En esta última sección se realiza el ciclo de entrenamiento principal. Al principio, se restablece el entorno y se inicializa el estado. Luego, se prueba una acción, se ejecuta, se observa la siguiente pantalla y la recompensa, y se optimiza el modelo una vez. Cuando termina el episodio (el modelo falla), se reinicia el ciclo. Un detalle importante es que hasta que la memoria no está lo suficientemente llena no empieza a trabajar

c. Despliegue

El siguiente capítulo recoge las diferentes plataformas seleccionadas para el despliegue del resultado del proyecto.

En la documentación presente se ofrecen las dos plataformas que se han utilizado en la ejecución: la primera y más utilizada, por medio de servidores físicos configurados para la ejecución de algoritmos de aprendizaje automático; y la segunda, su despliegue en la nube utilizando servidores virtuales.

d. Despliegue Local

En el desarrollo de aprendizaje por refuerzo profundo, hay dos componentes que resaltan en requerimientos de hardware para una mejor ejecución de los algoritmos: el procesador y la tarjeta gráfica. Es por ello por lo que se ha tenido en cuenta estos factores y las características del servidor local son:

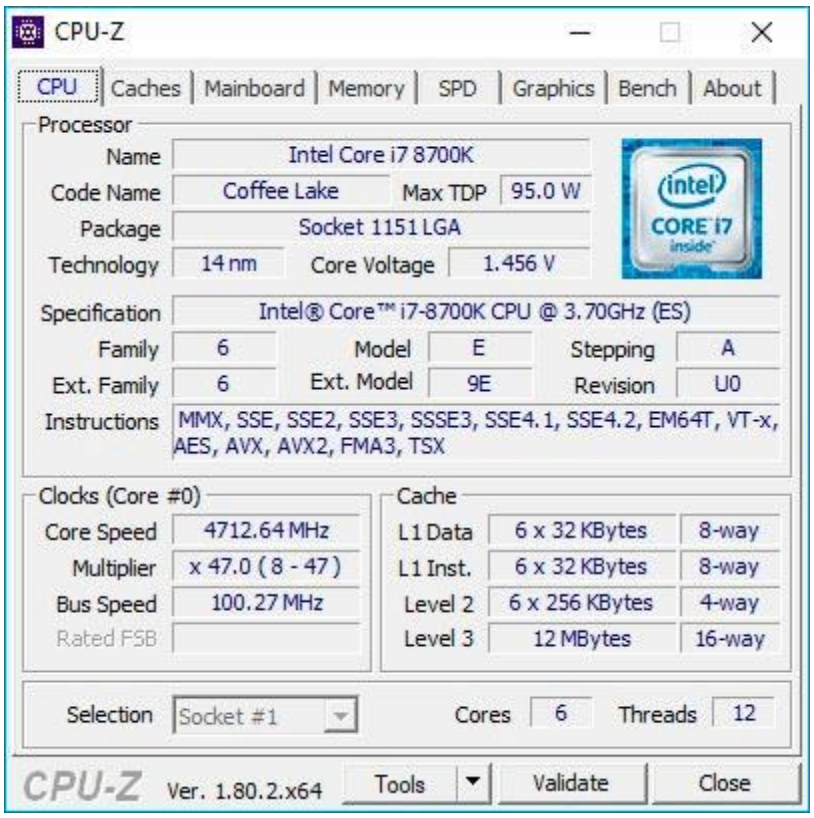


Ilustración 21. Características CPU - Servidor Local

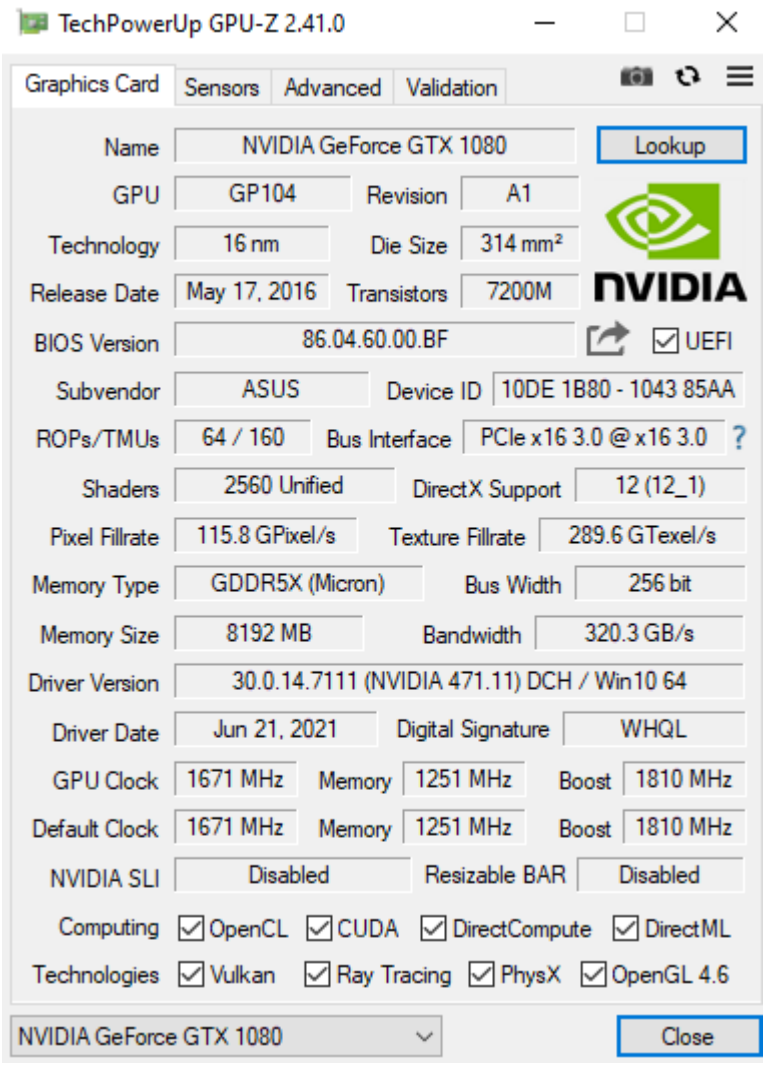


Ilustración 22. Características Tarjeta Gráfica - Servidor Local

Para la selección de la tarjeta gráfica hay que destacar que, en sus inicios, las GPU estaban dedicadas única y exclusivamente a los campos de la visualización, renderizado y edición de imágenes y videos. Con la evolución de las tecnologías y la inteligencia artificial, estás unidades han cobrado protagonismo en la capacidad de cómputo debido a la potencia que poseen actualmente. Todo esto ha permitido un avance en el rendimiento, ya que la paralelización de los cálculos en los algoritmos de aprendizaje automático es un salto importante.

El proyecto de investigación se ha realizado con el soporte de este avance y se ha aprovechado el paquete CUDA (Compute Unified Device Architecture) para realizar la

computación paralela, que mejora el rendimiento de la computación al utilizar las funciones de la unidad de procesamiento de gráficos (GPU).

Las características del servidor físico combinando el hardware con el paquete de CUDA para el soporte de cálculos de la GPU junto con la CPU permite la ejecución de algoritmos de aprendizaje por refuerzo profundo con un buen resultado.

e. Despliegue Virtual

Como alternativa al despliegue local en el servidor físico, se prueba el despliegue y ejecución de entornos en la nube, para el caso en el que se necesite potencias extra o el renderizado local no funcione correctamente. Para el despliegue comentado, se seleccionó la interfaz web "Google Colab", que permite acceder a servidores virtuales para escalar las necesidades de computación requeridas. Además, el servicio es gratuito, solo requiere estar registrado y ofrece GPU dedicadas que son potentes.

Se puede observar que Google Colab ofrece el servicio de la tarjeta gráfica "Tesla K80", que tiene las siguientes características:

```
!nvidia-smi
```

```
Wed Aug 11 15:59:44 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.42.01    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   70C    P8    35W / 149W |      0MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

Ilustración 23. Características de la tarjeta gráfica Tesla K80.

Ilustración 24. Características del despliegue virtual ofrecido en Google Colab.

f. Otros Servicios

El despliegue en nube tiene un amplio abanico de posibilidades debido a la gran extensión hoy en día de grandes cantidades de cómputo. Por ejemplo, existen los servicios de Microsoft Azure o los AWS que ofrecen la utilización de servidores virtuales. La selección entre ellos se realiza según el usuario, ya que las características ofrecidas son similares.

5. Experimentación

a. Cartpole

Este apartado incluye ejemplos de entrenamiento realizados en el entorno CartPole. Este es un problema que ha servido para la investigación en el campo del Aprendizaje por Refuerzo desde prácticamente sus inicios en el mundo tecnológico moderno. También conocido como Péndulo Invertido, es un péndulo con un centro de gravedad sobre su punto de pivote. Es inestable, pero se puede controlar moviendo el punto de pivote debajo del centro de masa. El objetivo final en el entorno es mantener el carro equilibrado aplicando las fuerzas adecuadas a su punto de pivote.

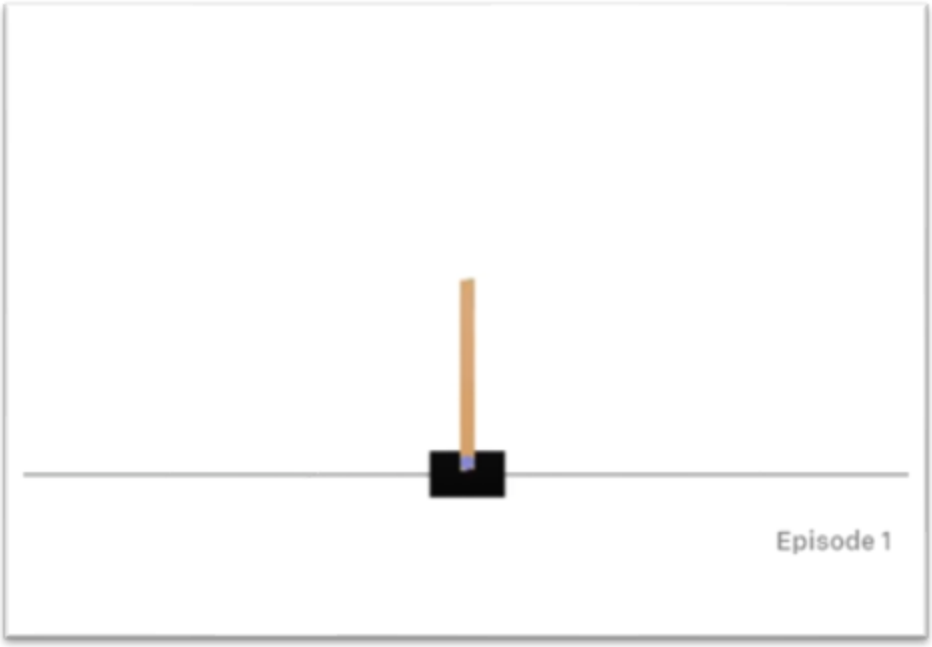


Ilustración 25. Ejemplo de la pantalla en el entorno Cartpole-v0.

Por lo tanto, el agente debe decidir entre dos acciones: mover el carro hacia la izquierda o hacia la derecha para que el poste conectado a él permanezca vertical.

```
Observation:
    Type: Box(4)
    Num     Observation               Min                     Max
    0       Cart Position             -4.8                    4.8
    1       Cart Velocity             -Inf                    Inf
    2       Pole Angle                -0.418 rad (-24 deg)    0.418 rad (24 deg
    3       Pole Angular Velocity     -Inf                    Inf

Actions:
    Type: Discrete(2)
    Num   Action
    0     Push cart to the left
    1     Push cart to the right
```

Ilustración 26. Espacio de observación y acciones del entorno CartPole-v0.

Cuando el agente observa el estado actual del entorno y selecciona una acción, el entorno pasa a un nuevo estado y devuelve una recompensa que indica las consecuencias de la acción. En este ejemplo, la recompensa por cada paso de tiempo incremental es +1, y si la barra cae demasiado o el carro está a más de 2.4 unidades del centro, el entorno termina. Esto significa que los escenarios de mejor desempeño se ejecutarán durante un período de tiempo más largo, acumulando una mayor rentabilidad.

La tarea CartPole está diseñada para que la entrada del agente sean 4 números reales. Cada estado del entorno representado por un valor. Sin embargo, la red neuronal puede resolver la tarea simplemente mirando la escena, por lo que el marco de la pantalla centrado en el carrito de compras se ha utilizado como entrada.

Se presenta el estado conseguido como la diferencia entre el cuadro de pantalla actual y el anterior. Esto permitirá al agente para tener en cuenta la velocidad del palo a partir de una imagen.

xx. Aplicación DQN

Para la experimentación con el entorno CartPole, se utilizó el algoritmo DQN. A continuación, se refleja un análisis del entrenamiento del agente mediante gráficas de los indicadores principales:

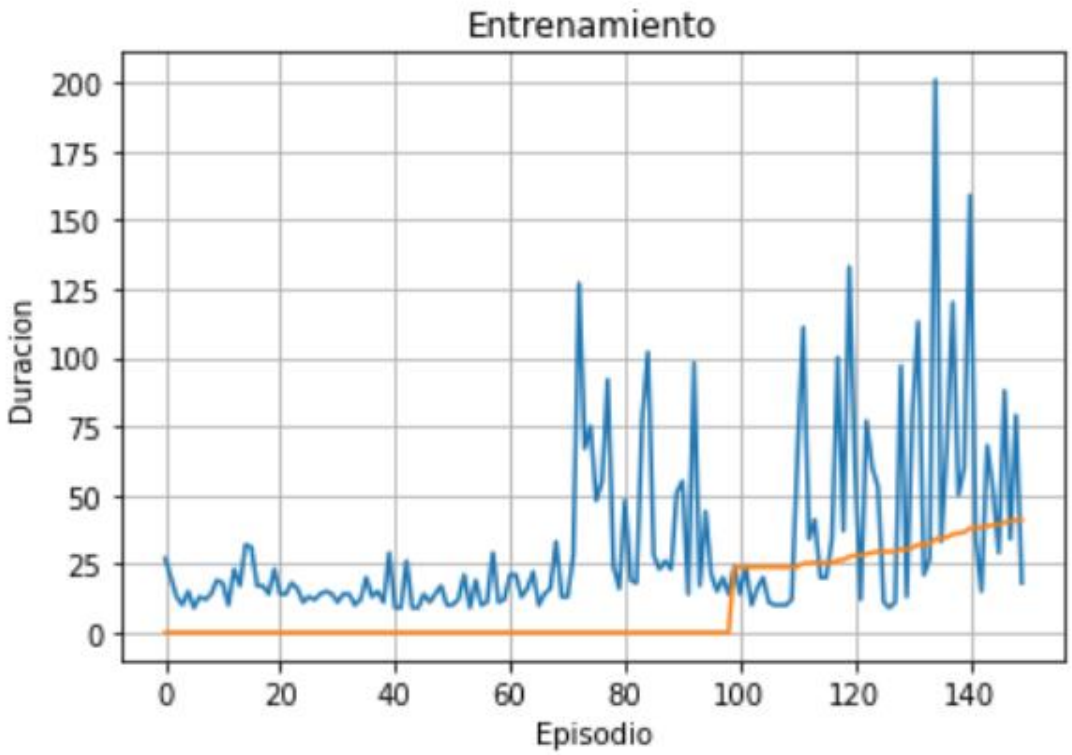


Ilustración 27. Ejemplo del resultado con algoritmo DQN - CartPole-v0.

b. Car 2D

La experimentación realizada en el entorno de CarRacing-v0 comparte las secciones de implementación utilizadas en el entorno de CartPole, pero varía en ciertas características modificadas para el correcto funcionamiento en este entorno. Además, en este apartado se ha implementado una modificación del algoritmo DQN (DDQN), ya que DQN tiende a ser más optimista en comparación con DDQN (más adelante se amplía este razonamiento).

xxi. Espacio de acción

El entorno CarRacing-v0 se compone de un espacio de acción continuo, por lo que se ha decidido transformarlo a un espacio con acciones discretas donde la búsqueda y la optimización mejora considerablemente. Para ello se han discretizado las acciones, limitándolas a cinco posibles acciones; izquierda, derecha, frenar, acelerar, nada. La acción *nada* permite que el agente mantenga la conducción en línea recta con velocidad constante.

La forma de pasar el espacio de direcciones a discreto utilizada ha sido un mapeo directo de las acciones discretas a continuas [14]:

Acción discreta		Acción continua
`Turn_left`	→	`[ -1.0, 0.0, 0.0 ]`
`Turn_right`	→	`[ +1.0, 0.0, 0.0 ]`
`Brake`	→	`[ 0.0, 0.0, 0.8 ]`
`Accelerate`	→	`[ 0.0, 1.0, 0.8 ]`
`Do-Nothing`	→	`[ 0.0, 0.0, 0.0 ]`

Ilustración 28. Mapeo del espacio de acciones.

Este tipo de discretización tan solo permite una acción continua por cada acción discreta, por lo que han surgido algunos problemas. Por ejemplo, que el modelo no pueda orientar la conducción y acelerar al mismo tiempo, con lo que al tomar las curvas no se puede frenar y girar a la vez, provocando en el modelo problemas como girar a baja velocidad.

La solución utilizada para modificar estos errores de comportamiento ha sido la discretización suavizada, donde los puntos discretos alrededor del espacio continuo de acciones

se encuentran localizados no solo cerca de las esquinas. Se trata de una pequeña modificación que manteniendo el espacio de cinco acciones final da buenos resultados.

xxii. Espacio de observación

Como se ha definido previamente en la documentación del proyecto, el espacio de observación del entorno de CarRacing-v0 es un marco RGB de 96x96 píxeles. El marco se forma a partir de una barra de control en la parte inferior de la pantalla, donde se visualiza la información referente al agente (dirección, fuerzas laterales, puntuación, etc.) y el mapa de simulación.

- El problema destacable es que la imagen es pequeña y un porcentaje alto de la información no se aprecia con claridad, por lo que se decidió realizar algunos cambios para mejorar el funcionamiento de la implementación: el marco RGB se transformó a un marco en escala de grises y se quitó el panel inferior, ya que no tenía gran relevancia.

xxiii. DQN a DDQN

Se decidió implementar DDQN después de probar DQN porque DDQN tiende a ser más escéptico con las acciones elegidas, es decir, a la hora de calcula el valor de Q objetivo para realizar la acción en cuestión. De forma general, DDQN ayuda a reducir la sobreestimación de los valores q, lo que permite minimizar el tiempo de entrenamiento y tener un aprendizaje con mayor porcentaje de estabilidad.

xxiv. Otras implementaciones

Alguno de los cambios añadidos es recortar la recompensa a un máximo de uno por paso. Con esto se consigue evitar en gran parte que el agente reciba muchos estímulos para alcanzar velocidades elevadas, lo que provoca la pérdida de control en curvas cerradas.

También se ha añadido un tiempo de "time out" para cuando el automóvil sale de la pista. El objetivo de este tiempo es minimizar las pérdidas de cálculo en los entornos, dado que cuando el automóvil se aleja de la pista se considera que está localizado en una posición no deseada, por lo que no agrega valor y no interesa seguir con la reproducción del episodio.

El resto de los cambios son propios de las pruebas en la configuración de los algoritmos. Desde los cambios en la arquitectura de las capas de las redes hasta los cambios de los hiperparámetros del modelo, como por ejemplo el decaimiento de épsilon, el valor del ratio de aprendizaje, etc.

xxv. Resultados

En la experimentación realizada se han utilizado diferentes configuraciones de los parámetros y números de episodios a través de un modelo personalizado seleccionado mediante las pruebas realizadas.

Los resultados obtenidos con el algoritmo DDQN son los siguientes:

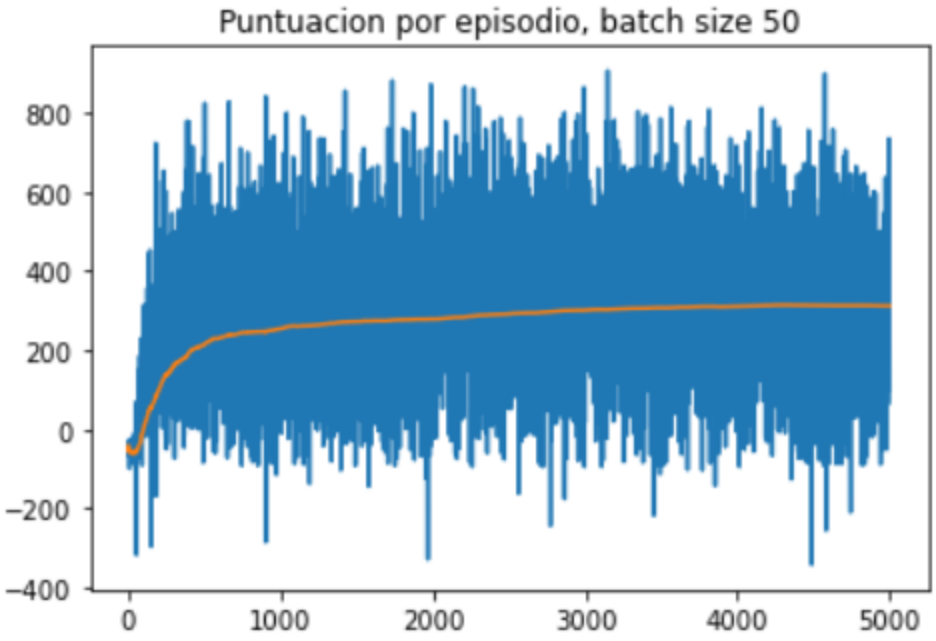


Ilustración 29. Duración por episodios de CarRacing-v0 con DDQN.

A simple vista se aprecia una cantidad significativa de fluctuación en la duración y en el aprendizaje. Las recompensas suben y bajan, pero la tendencia se mantiene levemente al alza con el paso de los episodios.

La puntuación objetivo de 800-900 es conseguida antes del episodio 1000, por lo que la curva de la media de duración se estabiliza más a partir de ese número de episodios. Esto se ha conseguido probando diferentes configuraciones de los hiperparámetros de la DDQN. Por ejemplo, variando los valores que; definen la política de épsilon, marcan el tamaño de la bolsa de ejemplos, asignan el ratio de aprendizaje… la capacitación y el aprendizaje se han visto mejorados.

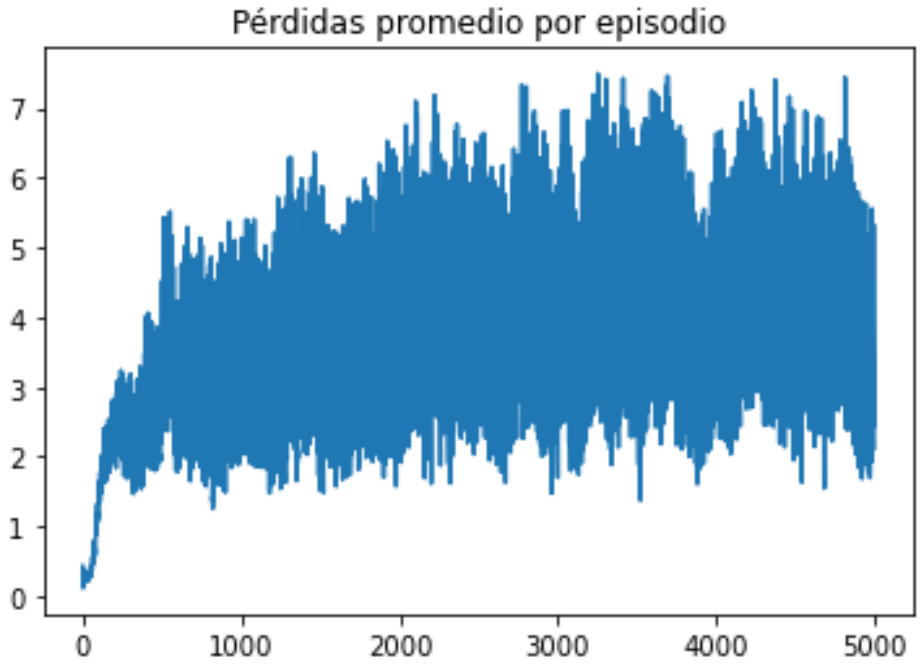


Ilustración 30. Pérdidas promedio por episodio de CarRacing-v0 con DDQN.

La pérdida se mide como la diferencia entre el resultado previsto y el real, y a lo largo de los episodios alcanza un valor máximo entorno al 7. En el Deep Learning, se minimiza el error estimado por la función de pérdida optimizando los pesos, pero en este caso el los ofrecido por

la red objetivo va aumentando. Esto no quiere decir que las pérdidas en la experimentación no se minimicen, ya que realmente lo hace. Este aumento de las pérdidas promedio se produce por el hecho de que al principio los valores que forman posteriormente el input de la red se escogen de forma aleatoria.

Las pérdidas se calculan conforme a la fórmula ofrecida en la ilustración 11 y se van estabilizando en el desarrollo de los episodios. Una forma de solucionar esta desviación prematura de las pérdidas sería congelar la red objetivo un intervalo de episodios mayor, por ejemplo, 500 episodios (La gráfica de la Ilustración 28 muestra el resultado congelando la red 1 episodio). De esta forma no se produciría un cambio tan brusco y se favorecería el aprendizaje levemente.

6. Conclusiones

Los primeros objetivos impuestos para este proyecto eran aprender a utilizar la herramienta de PyTorch, con la que nunca se había practicado, y trabajar con ella a través de CUDA, utilizando la Suite de código abierto, Anaconda. Estos se han logrado a alcanzar realizando un curso introductorio al Aprendizaje Profundo mediante el empleo de PyTorch previamente a empezar el proyecto. Con ello, se puede decir que los conocimientos previos a la realización de la investigación en materia software no son triviales y hay que tener una base amplia de entendimiento para poder comenzar con la experimentación a través de los algoritmos.

También, gracias a la información y ejemplos ofrecidos en internet, se ha logrado aprender el pensamiento que hay por detrás del Aprendizaje por Refuerzo Profundo, y como ha ido evolucionando en los últimos años en base a los análisis de los diferentes resultados que se han ido consiguiendo en las investigaciones.

Sobre la experimentación realizada, en general, se ha conseguido mediante los algoritmos DQN Y DDQN encontrar un aprendizaje correcto para el agente en los entornos

utilizados. Primero se ha obtenido la forma de que el agente en el entorno de CartPole mantenga el equilibrio un tiempo aceptable. Después, tras la consecución del objetivo en el primer entorno, se ha modificado el algoritmo para poder trabajar en el aprendizaje de CarRacing-v0, donde en un primer instante el algoritmo utilizado fue DQN. Posteriormente tras el análisis de los resultados se decidió aplicar DDQN para conseguir el objetivo previsto, ya que el algoritmo DQN en este entorno no ofrecía unos resultados aceptables en calidad de tiempo y cálculo.

Tras la investigación realizada, los diferentes entornos implementados y el examen y estudio de los resultados obtenidos, se han recopilado una serie de conclusiones, principalmente sobre el Aprendizaje por Refuerzo Profundo, pero también sobre el estudio de la IA. La gran variedad de aplicaciones de la IA, y más concretamente, del Aprendizaje por Refuerzo Profundo, es verdaderamente amplio. El avance en esta rama del conocimiento está siendo muy rápido y está cambiando el mundo como lo conocemos.

Respecto al desarrollo del código y su simulación, cabe resaltar la complejidad que se encuentra detrás de todo. Desde la inmensidad de librerías que se encuentran en Python en favor de la programación de IA hasta la optimización del hardware a través de paquetes para poder realizar todos los cálculos necesarios en el menor tiempo posible. Es un trabajo que necesita un mundo tecnológico en la sombra para poder funcionar correctamente desde un ordenador comercial.

La conclusión general de este trabajo de investigación es que no existe una solución óptima para todo y menos en el ámbito del Aprendizaje por Refuerzo Profundo, ya que dependiendo del entorno que se quiera abordar, una combinación diferente de; algoritmos, técnicas y muchos más factores puede funcionar mejor que en otros entornos. También, se puede agregar, que en este campo los recursos son muy importantes para el entrenamiento del agente y tener una capacidad de cómputo y tiempo sin límites sería lo óptimo.

7. Trabajos futuros

En este capítulo se realiza una recolección de diferentes oportunidades de investigación en un desarrollo futuro. El objetivo de estas trataría de alcanzar una mejora del trabajo expuesto a lo largo de los capítulos a través de algoritmos y técnicas diversas con los cambios necesarios.

a. Algoritmos

En la totalidad del trabajo de investigación se ha enfocado la implementación los algoritmos DQN y DDQN, que son algoritmos basados en Aprendizaje por Refuerzo Profundo. Es decir, forman parte de un conjunto de algoritmos más grande donde existen diferentes tipos que parten de la misma idea.

Existe otra evolución de DQN también, **Dueling DQN [15]**. En este algoritmo los valores Q se dividen en dos secciones diferentes: la función de valor y la función de ventaja. Este algoritmo propone que la propia red neuronal divida la última capa en dos partes para calcular los valores de las funciones, y como resultado, combinarlas para estimar los valores Q.

En la actualidad se están desarrollando nuevos algoritmos enfocados en esta rama de conocimiento. Por ejemplo, el **control episódico neuronal [16]** ha ofrecido buenos resultados en pruebas realizadas en una colección de juegos de tipo Atari.

Hay más algoritmos destacables en el campo del aprendizaje por refuerzo profundo que se podrían estudiar en trabajos futuros y poner a prueba con los entornos utilizados como **ACER**, **PPO2**, **TRPO** [17].

b. Técnicas

La finalidad del proyecto era la aplicación de Aprendizaje por Refuerzo Profundo a través del algoritmo DQN en algunos entornos de OpenAI Gym, pero, en varios de ellos surgen problemas con su empleo dado que en aquellos en los que la obtención de recompensa es muy difícil o se retrasa mucho en el tiempo el algoritmo no consigue alcanzar un sistema entrenado con éxito, por ejemplo, MountainCar-v0.

La solución en este tipo de entornos podría basarse en otro tipo de técnicas adicionales para conseguir llegar a un sistema entrenado con éxito. Algunas de estas técnicas nombradas podrían ser:

- **Aprendizaje supervisado [19]**: con el objetivo de mejorar la fase de exploración, se podría aplicar en un intervalo inicial el Aprendizaje Supervisado para comenzar el entrenamiento analizando soluciones ya obtenidas. De esta forma el proceso de exploración se habría simplificado y el problema para llegar a las recompensas dicho previamente sería abordado con menor dificultad.
- **Búsqueda en árbol [19]**: dentro de la gran variedad de entornos aportados por la plataforma de OpenAI Gym se encuentran en el entrenamiento de parte de ellos una cantidad muy amplia de combinaciones de estados posibles, donde es complicado que el agente obtenga recompensas en el proceso de explotación. Para ello es posible reunir los sistemas de Aprendizaje por Refuerzo y los de Búsqueda en Árbol. Estos último, por su cuenta es difícil que encuentren soluciones en tiempos viables, pero combinados con el Aprendizaje por refuerzo pueden llegar a conseguirlo.

c. Optimización automatizada de hiperparámetros

Se pueden agregar sistemas automáticos de optimización de hiperparámetros que reduzcan el sesgo producido al realizar la optimización de forma manual y realicen una búsqueda más

sistemática. Para ello se podría insertar diferentes métodos de búsqueda; en cuadrícula, bayesiana, aleatoria, etc.

d. Ejecución distribuida

La ejecución de los experimentos se inicia manualmente en el sistema que ejecuta. Por la cantidad de procesado necesario y el tiempo que toman los experimentos en ejecutarse, es interesante que se pueda decidir el sistema en el que se pueden ejecutar los experimentos, dependiendo de la carga de cada uno de los servidores disponibles; o que, incluso, pudiese paralelizarse la ejecución de un experimento en concreto [20].

8. Bibliografía

[1] BootcampAI.org. *Reinforcement Learning — Aprendizaje por refuerzo*. Reinforcement Learning — Aprendizaje por refuerzo

[2] Cui, Y., Huang, X., Wu, D., & Zheng, H. (2020). Machine Learning based Resource Allocation Strategy for Network Slicing in Vehicular Networks. *2020 IEEE/CIC International Conference on Communications in China, ICCC 2020*, *292*, 454–459. https://doi.org/10.1109/ICCC49849.2020.9238991

[3] Burbano Labrador, C., Gamboa, C., & W. Sanchez, C. (2017). *Desarrollo e Implementación de un Sistema Inalámbrico de Monitoreo de Variables Atmosféricas con Herramientas de Software y Hardware Libre*. https://www.researchgate.net/profile/Jhon_Fredy_Narvaez/publication/320170890_Desarrollos_de_la_Ingenieria_ambiental_en_la_evaluacion_de_la_calidad_de_los_recursos_naturales_y_la_salud_ambiental/links/59d26bfca6fdcc181ad611ce/Desarrollos-de-la-Ingenieria-

[4] Izaurieta, F., & Saavedra, C. (1999). Redes Neuronales Artificiales. *Charlas de Fisica*, 1–15. https://doi.org/10.1016/S0210-5691(05)74198-X

[5] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "A B7CEDGF HIB7PRQTSUDGQICWVYX HIB edCdSISIXvg5r ` CdQTw XvefCdS," *proc. IEEE*, 1998

[6] Mukkamala, Mahesh Chandra, Hein, Matthias, 34th International Conference on Machine Learning, ICML 2017, 3917-3932.

[7] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A.,

Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236

[8] *Markel Sanz.* https://markelsanz14.medium.com/introducci%C3%B3n-al-aprendizaje-por-refuerzo-parte-3-q-learning-con-redes-neuronales-algoritmo-dqn-bfe02b37017f

[9] *Anaconda*. https://blog.desdelinux.net/ciencia-de-datos-con-python/

[10] Cleverpy.com. *PyTorch: ¿Qué es?* https://cleverpy.com/que-es-pytorch-y-como-se-instala/

[11] *CUDA*. https://www.profesionalreview.com/2018/10/09/que-son-nvidia-cuda-core/

[12] @AIMatesa.*DeepQ-Learning*. https://pbs.twimg.com/media/EVF2Y1AXsAApFF8?format=jpg&name=4096x4096

[13] Hasselt, H. Van, Guez, A., & Silver, D. (2016). Double DQN.pdf. *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, 2094–2100.

[14] Mike.W. (n.d.). *Solving Car Racing with Proximal Policy Optimisation*. https://notanymike.github.io/Solving-CarRacing/

[15] Zhao, Y., Wang, Z., Yin, K., Zhang, R., Huang, Z., & Wang, P. (2020). Dynamic reward-based dueling deep Dyna-Q: Robust policy learning in noisy environments. *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence*, 9676–9684. https://doi.org/10.1609/aaai.v34i05.6516

[16] Pritzel, A., Uria, B., Srinivasan, S., Badia, A. P., Vinyals, O., Hassabis, D., Wierstra, D., & Blundell, C. (2017). Neural episodic control. *34th International Conference on Machine Learning, ICML 2017*, *6*, 4320–4331.

[17] *PPO*. https://openai.com/blog/openai-baselines-ppo/

[18] Cang Ye, N. H. C. Yung and Danwei Wang, "A fuzzy controller with supervised learning assisted reinforcement learning algorithm for obstacle avoidance," in *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 33, no. 1, pp. 17-27, Feb. 2003, doi: 10.1109/TSMCB.2003.808179.

[19] Anthony, T., Tian, Z., & Barber, D. (2017). *Thinking Fast and Slow with Deep Learning and Tree Search*. http://arxiv.org/abs/1705.08439

[20] Akiba, T., Fukuda, K., & Suzuki, S. (n.d.). *ChainerMN: Scalable Distributed Deep Learning Framework *.*